Contents

- Halo: The Master Chief Collection
 - Halo: The Master Chief Collection Documentation Homepage
 - Excession Tool
 - Excession Overview
 - Initial Setup Home
 - Folder Structure
 - Configuring Map Folders
 - Creating a Mod Package Home
 - Campaign Mod Package
 - Multiplayer Maps Mod Package
 - Firefight Maps Mod Package
 - Specifying Properties
 - Viewing and Editing a Mod Package
 - Uploading a Mod Package
 - Appendix A - UI of Excession
 - Appendix B - multiplayer_object_types.bin file
 - Halo CE - Coming Soon
 - Halo 2
 - How-Tos
 - How-Tos Home
 - Build Cache (Map) Files
 - Create Effective Bitmaps
 - Dynamic Lights Best Practices
 - Fix "Creeping" Idle Animations
 - Information on material_effects
 - Set Instanced Geometry Pathfinding Policy
 - Set Object Reset Height in MP
 - Setting Up A Character With Basic AI
 - Set Up Controller Cheats

- Set Up King of the Hill in MP
- Set Up Teleporters in MP
- Split Scenario Resources
- Using screenshot_cubemap Command

AI

- AI Overview
- Behavior List
- Engineering Outline
- Pathfinding

Art

- Art Home
- Animation
 - Damage Animations
 - First Person Animations
 - Inherited Animations
- Bitmaps
 - Bitmaps Home
 - Bitmap Plates
 - Import Bitmaps into the Game
- Decorators
 - Decorators Home
 - Floating Decals
 - Models
 - Screen Facing Quads
- Decals Overview
- Crates Overview

Audio Overview

Devices

- Devices Home
- Device Machines
- Device Controls

HaloScript Reference

- Halo 2 Anniversary - Coming Soon
- Halo 3
 - Quick Start Overview
 - Content Creation Quick Start
 - Step 0 - Prerequisites
 - Step 1 - Modelling Level Geometry in 3D Tool
 - Step 2 - Exporting Geometry to FBX
 - Step 3 - Conversion from FBX to ASS
 - Step 4 - Creation of Initial Tag Files
 - Step 5 - Modification of Tags in Guerilla
 - Step 6 - Calculation of Lighting
 - Step 7 - Populating the Level in Sapien
 - Step 8 - Loading the Level from Tags
 - Step 9 - Creation of the MAP file
 - Designer Boot Camp
 - Section 1
 - Section 2
 - How-Tos
 - Build Cache Files
 - Create Slip Surfaces
 - Create Soft Ceilings
 - Executing Multiple Script Commands
 - Find Console Commands
 - Setup Init.txt
 - Init.txt Scenario Pruning
 - Multiplayer Map Setup
 - Using Portals
 - Water Physics
 - Weapon Overview
 - Tools Overview
 - Device Machines
 - Gameshell UI

- HaloScript Overview
- Guerilla
 - User Interface
 - Edit Menu
 - File Menu
 - Filters
 - Source Control Menu
 - View Menu
 - Window Menu
 - Color Picker
 - Help Menu
- Sapien
 - Overview
 - Game Window
 - Hierarchy View
 - Tool Window
 - Asset Manipulation Gizmo
 - Output Window
 - Placing Objects
 - Placing Lights
 - Properties Palette
 - Structure Painter

AI Overview

- Activities
- Brute Variations
- Character Variant Names
- Combat Status
- Debug Commands
- Difficulty Levels
- Exhaustion and Max Duration
- Leadership
- Objective Window Color Key

- Relaxation Poses
- Task Status
- Art Overview
 - 3DS Max
 - Bulk Import and Export
 - Maya Cinematic Texture Camera
 - Animation
 - Facial Animation
 - Vehicle Animation
 - Naming Conventions
 - Decals Overview
 - System
 - Decorators Overview
 - Modeling
 - Painting
 - Texturing
 - Objects Overview
 - Halo Script Damage_Object
 - Functions
 - Model Variants and Permutations
 - Setting Damage Sections and States
 - Setting Global Material Types
 - Setting Model Variants and Permutations
 - Setting Regions and Perutations
 - Tagging Damage Sections and States
 - Instance Objects Overview
 - Collision
 - Damage Sections and States Tag
 - Functions
 - Light Map Resolution
 - Pathfinding Policy
 - Shaders Overview

Albedo Properties

Albedo Blend Properties

Alpha Test Properties

Ambient Coefficient Properties

Ambient Tint Properties

Analytical Anti-Shadow Control Properties

Analytical Specular Contribution Properties

Area Specular Contribution

Blend Mode

Bump Mapping Properties

Diffuse Coefficient

Diffuse Tint

Double Multiply

Environment Map Coefficient

Environment Mapping

Environment Map Specular Contribution

Environment Map Tint

Environment Map Tint Color Properties

Fresnel Coefficient

Fresnel Curve Bias Properties

Fresnel Curve Steepness

Glancing Specular Power

Glancing Specular Tint Properties

Material Model Properties

Material Texture Properties

No Dynamic Lights

Normal Specular Power

Normal Specular Tint

Occlusion Parameter Map

Order 3 Area Specular

Parallax Properties

Rim Coefficient

Rim Maps Transition Ratio
Rim Power
Rim Start
Rim Tint
Roughness Properties
Self-Illumination Properties
Specular Coefficient Properties
Specular Map
Specular Mask Properties
Specular Power
Specular Tint
Standard Properties
Subsurface Coefficient
Subsurface Map
Subsurface Propagation Bias
Subsurface Tint
Templates
Terrain Overview
Transparence Coefficient
Transparence Map Properties
Transparence Normal Bias
Transparence Tint Properties
Dynamic Lights
BSP Overview
Debug Overview
Debug Menu
Profile Settings Commands
Misc Overview
Bitmap Curve Mode
Cheats.txt
Console Commands
Gravity Lifts

Optimal Texture Size Reference

Tool Bitmap Suffix Reference

Halo 3 ODST - Coming Soon

Halo Reach

Art

FBX to GR2

Halo 4 - Coming Soon

Known Issues

Excession Tool Overview

11/23/2022 • 2 minutes to read

Excession is one of the modding tools for Halo MCC.

This tool allows you to upload your mods for Halo MCC to the Steam Workshop. Particularly, Excession supports uploading of mods for the following Halo titles:

- Halo: Combat Evolved
- Halo 2: Classic
- Halo 2: Anniversary
- Halo 3
- Halo 3: ODST
- Halo: Reach
- Halo 4

Before uploading, you can perform all necessary configuration of your mod using this tool: you can create a campaign consisting of of multiple maps, add a set of multiplayer or Firefight maps, and so on. During this setup, you can configure all necessary images and logos, names, descriptions, map settings, add links to appropriate Game Variant files, configure insertion points, etc.

This guide provides basic info on the usage of this tool and operations necessary to configure a mod (after creation of its .map files) and upload it to Steam Workshop.

If you are looking for a specific section, you can find links to each area below:

Initial Setup

Creating a Mod Package

Viewing and Editing a Mod Package

Uploading a Mod Package

Appendix A: UI of Excession

Appendix B: How to generate the "multiplayer_object_types.bin" file

Initial Setup Home

11/23/2022 • 2 minutes to read

Folder Structure, Creating Root Folders, and ModInfo.json

Configuring Map Folders in Excession

Folder Structure, Creating Root Folders, and ModInfo.json

11/23/2022 • 3 minutes to read

First of all, we recommend you to put your mods to a single or multiple root folders.

For example, every root folder can correspond to a certain target Halo title your mods correspond to (e.g. "**Halo3**", "**Halo3 ODST**"). Or, they can correspond to multiple mod types (e.g. "**Campaigns**", "**Multiplayer Maps**"). Or, you can simply put all your mods to a certain single root folder (e.g. "**My Mods**").

Within these root folders, every mod will have a separate mod folder. For example: **...\My Mods\My Multiplayer Map 1**.

Within this mod folder, you will gather all the content of this particular mod. The subfolders and files within the mod folder may vary depending on the mod package and its content.

You can either create a mod folder beforehand and then select it in the **Mod Folder** field of your mod while creating it, or create a mod folder directly when creating a mod. However, you will not be able to change the mod folder after the creation of the mod. After the creation of the mod, the **Mod Folder** field will become read-only. So, please, choose the location for your mod carefully.

> **NOTE**
>
> **NOTE #1**: When the tool identifies and displays mods, it searches for them within the configured root folders. During this process, it assumes that all mod folders are contained directly within the root folder (i.e. not below the 1st level of folders within the root folder). If they located deeper in the folder hierarchy, the tool will not find them.

> **NOTE**
>
> **NOTE #2**: If you select a mod folder that is outside the root folder, its parent directory will be automatically added to the system as a new root folder, for the system to be able to find your mod.

All **.map** files of the mod package need to be manually placed to the **maps** subfolder of the mod folder.

> **NOTE**
>
> **NOTE #1**: When you will be specifying the properties of any map within your mod package, you will need to specify the **Scenario File Name** for it. Please note that in this field you need to specify ***neither*** the name of the **.map** file *nor* the path to it, ***but*** the path and name of the **.scenario** tag file that was used as a source for your **.map** file at the stage of its creation (e.g. by **tool.exe** or **tool_fast.exe**). This path needs to be specified relative to the **tags** folder and without the file extension of the **.scenario** tag file. For example, if you were compiling **.../tags/levels/my_campaign/my_campaign_map1.scenario** by **tool.exe** to obtain the **my_campaign_map1.map** file, then you will need to specify **levels/my_campaign/my_campaign_map1** as the value of the **Scenario File Name** field. See 3.6. Specifying Properties of a Map for details.

NOTE

NOTE #2: The current version of Excession does not check the existence of **.map** file or correctness of the **Scenario File Name** value. During the upload, Excession simply uploads all contents of the mod folder to Steam Workshop, without any validation of these aspects. So, please check the existence of all **.map** files and correctness of **Scenario File Name** values manually.

Moreover, it is convenient to store all images related to the mod package within the **images** subfolder. Along with that, some files within the mod folder will be created by Excession.

Particularly, for every mod package, Excession always creates the **ModInfo.json** file that stores all general info about the mod and is used for its identification by the tool. Other files within the folder depend on the type of the mod, e.g. for "multiplayer map" mods, the tool will create the **multiplayer** folder with a **.json** files containing the settings of added maps.

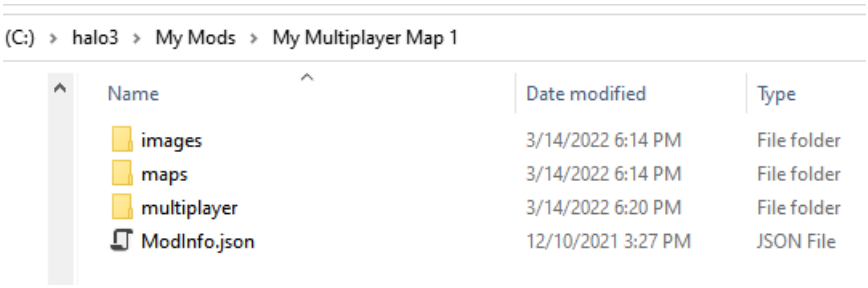


Fig.1. Mod folder with files of the mod of the "multiplayer map" type (after the creation of the mod)

Configuring Map Folder in Excession

11/23/2022 • 2 minutes to read

Initially, at the first launch of Excession, it will not display any list of the mods. At the **Home Screen** tab, it will display only three buttons: **CONFIGURE**, **REFRESH**, and **CREATE**.

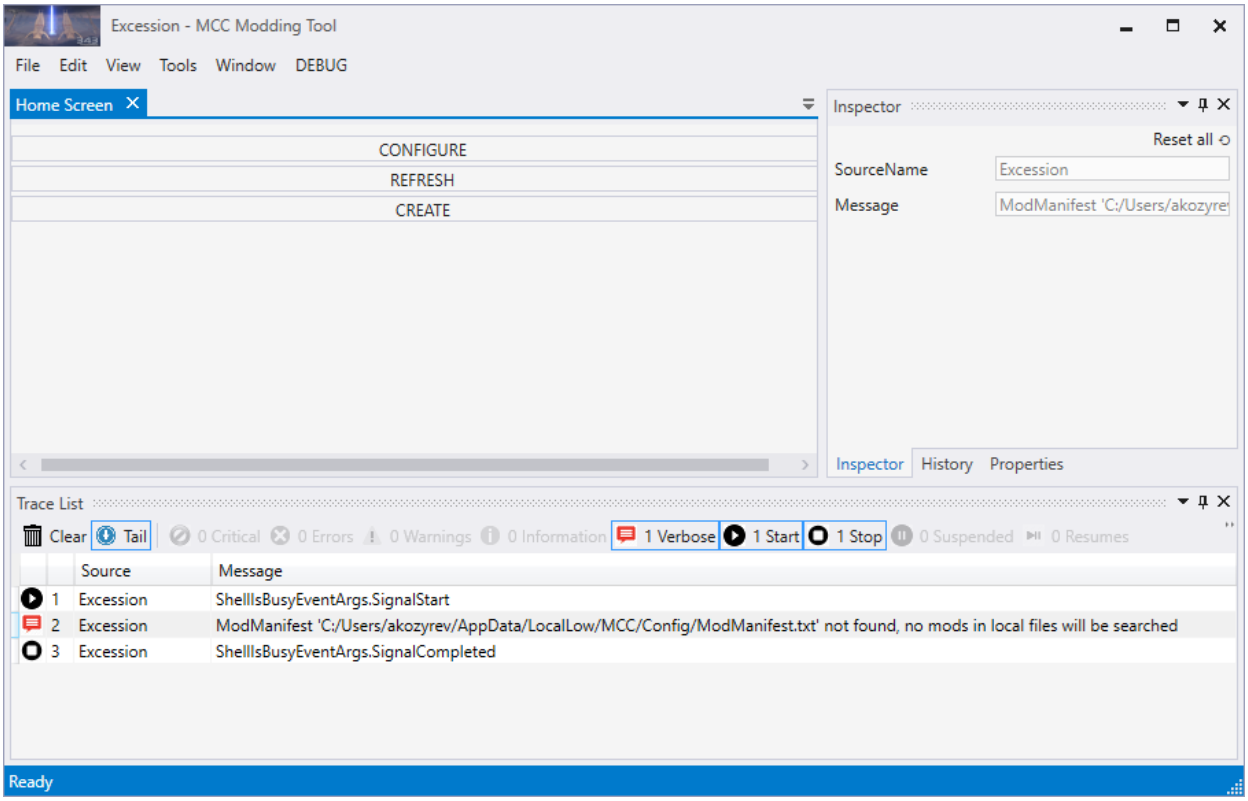


Fig.1. The UI of Excession at the first launch

The **CONFIGURE** button allows you to configure folders where Excession will look for mods.

You can configure them by doing the following:

1. Click **CONFIGURE**.
2. To add a root folder (see folder structure) with mods to Excession configuration, click **ADD FOLDER**.
3. After that the new list entry corresponding to a new root folder will appear. To assign it to particular root folder with mods, click **BROWSE** next to this list entry and select the target root folder in the appearing dialog. After that, the path to this folder will be displayed within this list entry.

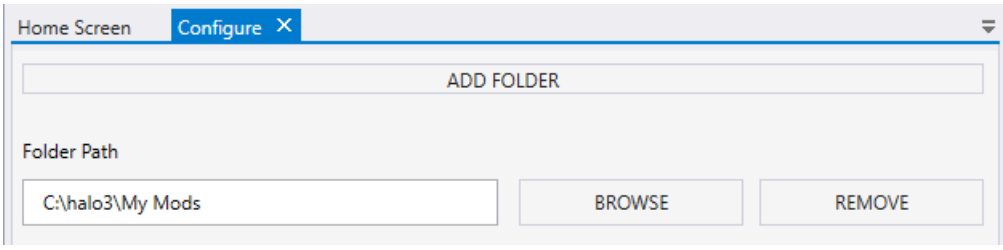


Fig.2. View of the configure option ui.

4. By clicking **ADD FOLDER** (and selection of the folder via **BROWSE**), you can add as many root folders as you like. For example, you can add a root folder for your mods for Halo 3 and a separate root folder for

your Halo CE mods.

5. If you want to remove some of root folders, click **REMOVE** next to them. To save your changes to the tool configuration, select **File > Save** or **File > Save All** from the main menu.

> **NOTE**
>
> **NOTE #1**: When the tool identifies and displays mods, it searches for them within the configured root folders. During this process, it assumes that all mod folders are contained directly within the root folder (i.e. not below the 1st level of folders within the root folder). If they located deeper in the folder hierarchy, the tool will not find them.

> **NOTE**
>
> **NOTE #2**: During the process of creating of a mod package, if you select a mod folder that is outside the root folder, its parent directory will be automatically added to the system as a new root folder, for the system to be able to find your mod.

Creating a Mod Package

11/23/2022 • 3 minutes to read

After configuring a root folder, you can start to create mod packages.

To do that:

1. At the Home Screen tab of Excession, click CREATE.
2. This will open the new MOD Info Editor tab.
3. At the top, this tab displays the Mod Folder field with a BROWSE button. This field allows you to select the folder for your mod. After clicking BROWSE, you will see the standard Windows dialog that allows you to select an existing folder or, if necessary, create a new folder and select it as the mod folder.

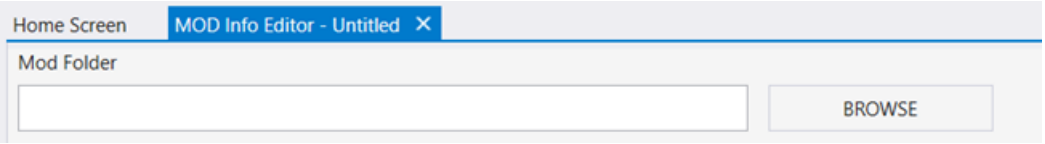


Fig 1. View of the Mod Info Editor UI.

> **WARNING**
>
> You will ***not*** be able to change the mod folder after the creation of the mod. After the creation of the mod, the **Mod Folder** field will become read-only. So, please, choose the location for your mod carefully.

> **NOTE**
>
> If you select a mod folder that is outside the root folder, its parent directory will be automatically added to the system as a new root folder, for the system to be able to find your mod.

4. Below this field, this tab displays the two fields with identifiers:
 - **Mod ID** – displays the identifier of the mod that is used by Excession and will be used by the target Halo title of the mod.
 - **Steam Workshop ID** – will be used to identify this mod package in Steam Workshop. At first, it is undefined, it will be displayed after upload of the mod package to Steam Workshop.

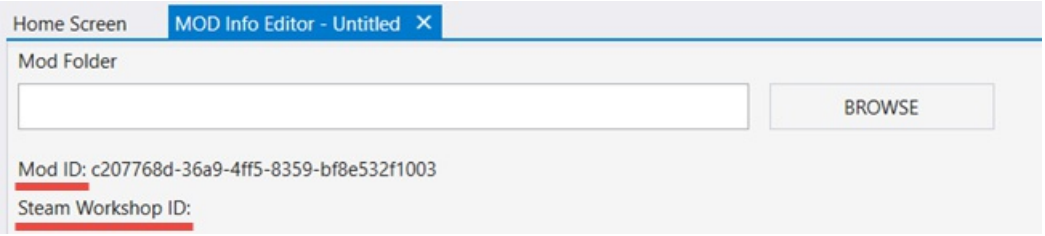


Fig 2. View of the identifiers within the Mod Info Editor UI.

5. To specify the mod properties, fill in other fields on the **MOD Info Editor** tab:

- **Title** – The title of your mod package.
- **Description** – The description of your mod package.
- **Mod Version** – The version of your mod in the following format:

<Major_version>.<Minor_version>.<Patch_version>. For example, **1.2.0**

- **Min. Game Version** – The minimum version of the *game* (target Halo title) required to use your mod, in the *<Major_version>.<Minor_version>.<Patch_version>* format.
- **Max. Game Version** – The maximum version of the game (target Halo title) that can be used with your mod, in the *<Major_version>.<Minor_version>.<Patch_version>* format.

> **NOTE**
>
> Values of the **Min. Game Version** and **Max. Game Version** fields will be used to check compatibility of the mod with the target game. Please note that correctness of these values will be checked during the identification of the mod (after its creation) by Excession and, also, using the same means, by the game itself. In case of the incorrect values, you will not receive any error during the mod creation itself, but you will receive the "***Mod does not support this version of the product (app)***" error by Excession during the identification of the mod (after **REFRESH**).

- **Game Title (Optional)** – The target Halo title of the mod (e.g. "Halo 3" for a mod targeted at Halo 3).

> **NOTE**
>
> **NOTE #1**: **The Game Title (Optional)** field is optional. I.e., you are able to either select a blank value in this drop-down (do not select any Halo title) or explicitly specify that your mod is not targeted at any Halo title by selecting "None" in this drop-down. The result will be the same – your mod will not be targeted at the particular Halo title, which can be useful for some mods.

> **NOTE**
>
> **NOTE #2**: Some properties of a mod are available only for mods targeted at the particular Halo title. These properties can be filled in only when this title is selected in the **Game Title (Optional)** field. Moreover, using the value of this field, the game itself will be able to identify that the mod is appropriate for it.

- **Campaign** section – This section allows you to add the content of the regular campaign to your mod, see creating a "Campaign" Mod Package for details.
- **Multiplayer Maps** section – This section allows you add multiplayer maps to your mod, see creating a "Multiplayer Maps" Mod Package for details
- **Firefight Maps** section – This section allows you add Firefight maps to your mod, see creating a "Firefight Maps" Mod Package for details.

Additional information on specifying the properties of a map can be found here.

> **NOTE**
>
> You need to add content in one or more sections above (e.g. **Campaign**, **Multiplayer Maps**, etc.), since in these sections you will add the main content of the mod itself (maps, and so on).

- **Advanced panel** – This panel allows you to specify the advanced settings of the mod package.
- **Disabled** – No source components are required. I.e., this means that all shared files are expected to be local to the mod.

6. To save your changes, select **File > Save** or **File > Save All** from the main menu of Excession.

Creating a Campaign Mod Package

11/23/2022 • 2 minutes to read

During the creation of the mod package, you can add a campaign with multiple maps to it by clicking **CREATE** in the **Campaign** section.

> **NOTE**
> The mod package can contain only one campaign.

To create a "Campaign" mod, you need to do the following

1. In the **MOD Info Editor** tab of the mod package, click **CREATE** in the **Campaign** section.
2. This will open the new **Campaign** tab where you can specify properties of the campaign. The **Campaign GUID** field in this tab will display the automatically generated identifier of the campaign

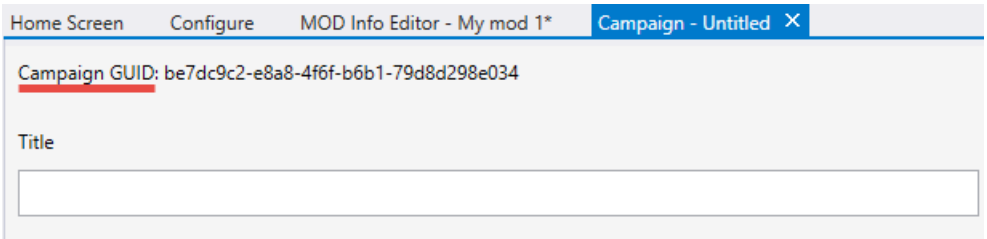


Fig 1. View of the Campaign GUID.

3. To specify properties of the campaign, fill in other fields on the **Campaign** tab:
 - **Title** – The title of the campaign.
 - **Description** – The description of the campaign.
 - **Campaign Missions** section – This section allows you to add maps to the created campaign. You can add as many maps as necessary. Adding a map is performed in a unified way, using a new **Map** tab, see Specifying Properties of a Map for details. Along with regular properties that are the same for maps of all types, your **Map** tab will also contain additional properties actual for campaign maps only, see Campaign Mission (Map) properties. After the properties of the map are saved in the **<name_of_scenario>.json** file within the **campaign** subfolder, this map will appear in the list the **Campaign** tab:

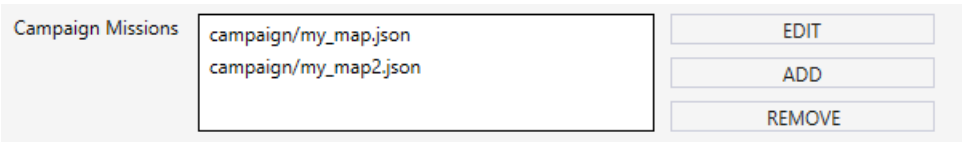


Fig 2. View of the current campaign missions.

> **NOTE**
> If you add multiple maps to the list, you can change their order using up and down arrows displayed next to the list. (Currently, they are in development and can be displayed not for all lists.)

 - After adding a map, if necessary, you can edit its properties. To do that, select it in the list and click

EDIT. Or, you can remove the map from the campaign (select it and click **REMOVE**).

4. To save your changes, select **File** > **Save** or **File** > **Save All** from the main menu of Excession.

If necessary, you can remove the campaign from the mod package. To do that, click **CLEAR** next to the **Campaign** section (in the **MOD Info Editor** tab of your mod).

Creating a Multiplayer Maps Mod Package

11/23/2022 • 2 minutes to read

You can add a single or multiple multiplayer maps to your mod package.

> **NOTE**
> The number of multiplayer maps that you can add to a mod package is not limited.

To do this:

1. In the **MOD Info Editor** tab of the mod package, click **CREATE** in the **Multiplayer Maps** section.

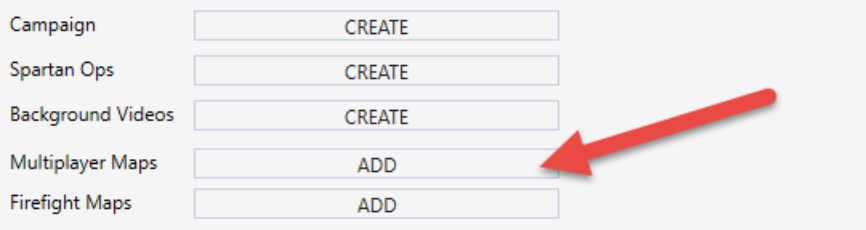


Fig 1. View of the Multiplayer Maps button in the Mod Info Editor.

2. This will open the new **Map** tab where you can specify properties of the multiplayer map that will be added to your mod package. Adding a map is performed in a unified way, see Specifying Properties of a Map. Along with regular properties that are the same for maps of all types, your **Map** tab will also contain additional properties actually for multiplayer maps only, see Multiplayer Map properties. After the properties of the map are saved in the **<name_of_scenario>.json** file within the **multiplayer** subfolder, this map will appear in the list within the **Multiplayer Maps** section.

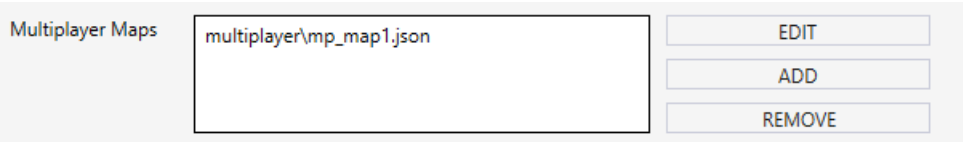


Fig 2. View of the Multiplayer Maps currently included.

> **NOTE**
> If you add multiple maps to the list, you can change their order using up and down arrows displayed next to the list. (Currently, they are in development and can be displayed not for all lists.)

After a multiplayer map, if necessary, you can edit its properties. To do that, select it in the list and click **EDIT**. Or, you can remove the map from the mod package by selecting it and clicking **REMOVE**. To add another map to the mod package, click **ADD** again and repeat the procedure for it.

3. To save your changes, select **File** > **Save** or **File** > **Save All** from the main menu of Excession.

Creating a Firefight Maps Mod Package

11/23/2022 • 2 minutes to read

You can add Firefight maps to your mod package.

> **NOTE**
> The number of Firefight maps that you can add to a mod package is not limited.

To do this:

1. In the **MOD Info Editor** tab of the mod package, click **CREATE** in the **Firefight Maps** section.

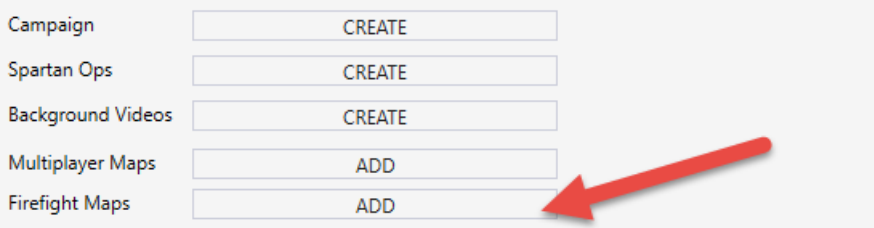


Fig 1. View of the Firefight Maps button in the Mod Info Editor.

2. This will open the new **Map** tab where you can specify properties of the multiplayer map that will be added to your mod package. Adding a map is performed in a unified way, see Specifying Properties of a Map. Along with regular properties that are the same for maps of all types, your **Map** tab will also contain additional properties actually for Firefight maps only, see Fireight Map properties. After the properties of the map are saved in the **<name_of_scenario>.json** file within the **firefight** subfolder, this map will appear in the list within the **Firefight Maps** section.

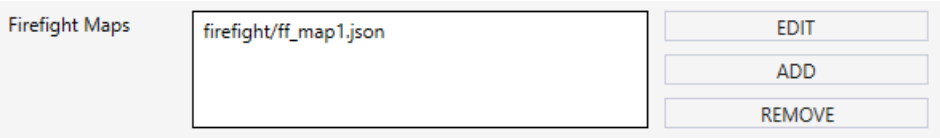


Fig 2. View of the Firefight Maps currently included.

> **NOTE**
> If you add multiple maps to the list, you can change their order using up and down arrows displayed next to the list. (Currently, they are in development and can be displayed not for all lists.)

After a multiplayer map, if necessary, you can edit its properties. To do that, select it in the list and click **EDIT**. Or, you can remove the map from the mod package by selecting it and clicking **REMOVE**. To add another map to the mod package, click **ADD** again and repeat the procedure for it.

3. To save your changes, select **File** > **Save** or **File** > **Save All** from the main menu of Excession.

Specifying Properties of a Map

11/23/2022 • 13 minutes to read

When you add a map to a mod package (e.g. as a campaign map; or, as a multiplayer map; etc.) you do it in a unified way.

Particularly, when you initiate this process, the tool opens a new **Map** tab where you can specify map properties.

Most of the main properties of the map are the same regardless of its type. However, some properties do differ and are specific for maps of particular type.

Two fields at the top of the **Map** tab are read-only. Particularly, you cannot change:

- **Map GUID** – The automatically generated identifier of the map.
- **Map Type** – The type of the map. This type is also defined automatically, based on whether you are adding this map as a campaign map, or as a multiplayer map, or as a Firefight map.

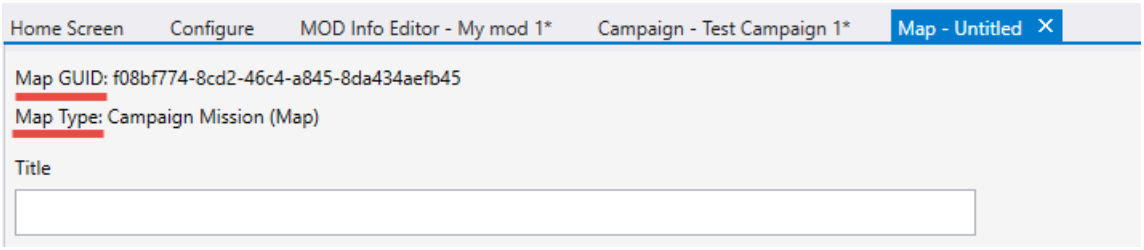


Fig 1. View of the Multiplayer Maps button in the Mod Info Editor.

Below go the main properties of the map that are the same for all types of the maps:

- **Title** – The title of the map.
- **Description** – The description of the map.
- **Scenario File Name** – The name and path to the of the ***.scenario** tag file* that was compiled to obtain the **.map** file of the level. This path needs to be specified relative to the **tags** folder (where all tag files are located in the Halo Editing Kit), without the **.scenario** file extension. Please note that the value of this field is neither the name of the **.map** file itself nor the path to it, it is the name and path to the **.scenario** tag file used for the **.map** file creation, and this difference is important.
 - Let's illustrate this difference and the process itself of an example:
 - During the process of your level creation (i.e. the creation of its tag files), you have created the **.../tags/levels/my_campaign/my_campaign_map1.scenario** tag file. It is the main tag file of your level.
 - After you have finished the level creation, you compile it (e.g using **tool.exe** or **tool_fast.exe**) to the **.map** file. After the compilation, you receive the **my_campaign_map1.map** file.
 - When you are creating a mod package in Excession and adding this map to it, you need to:
 - Put the **my_campaign_map1.map** file to the **maps** subfolder of your mod folder.
 - In Excession, when specifying properties of this map in the **Map** tab, specify "**levels/my_campaign/my_campaign_map1**" (without quotes) as the value of the

Scenario File Name field.

> **NOTE**
>
> The current version of Excession does not check the existence of .map file or correctness of the Scenario File Name value. During the upload, Excession simply uploads all contents of the mod folder to Steam Workshop, without any validation of these aspects. So, please check the existence of all .map files and correctness of Scenario File Name values manually.

Below, you can add some images to the map properties:

- **Image for Thumbnail** – The image that will be used as thumbnail of the map. The recommended file size of this image is around 900KB. You can use the image below as a template for the thumbnail.



- **Large Image** – The image that will be used as the large image of the map. The recommended file size of this image is around 900KB. You can use the image below as a template for the large image.



- **Image for Loading Screen** – The image that will be shown as the loading screen for the map. The game needs to be able to display this image in the resolution of the game (when loading the map itself), and, because of that, it should be a high-resolution image (e.g. 1920x1080).

> **NOTE**
>
> **NOTE #1**: To create a correct mod package and include all images in it, all images related to the mod must be located within the mod folder. I.e., you are not permitted to use images from other folders – in this case you will receive an error. Typically, all images of the mod package are stored in the **images** subfolder of the mod folder.

> **NOTE**
>
> **NOTE #2**: All these images must be stored in the PNG (.png) format.

> **NOTE**
>
> **NOTE #3**: For multiplayer maps and Firefight maps, you can also add an **Image for Top Down View**, see below.

Below that, Excession displays the set of expandable panels with properties that are specific for particular map types:

- **Multiplayer Map**
- **Firefight Map**
- **Campaign Mission (Map)**

Depending on the type of the map, the corresponding panel will appear expanded (automatically), all others will be locked and closed. See descriptions of these panels in the subsections below:

Multiplayer Map properties

Firefight Map properties

Campaign Mission (Map) properties

Adding Insertion Points

To save your changes (after you have specified map properties), select File > Save or File > Save All from the main menu of Excession.

Multiplayer Map Properties

The **Multiplayer Map** panel allows you to specify the following properties of a multiplayer map:

- **Allow Saved Films** – This option enables or disables the ability of players to save videos of their games on this map.
- **Image for Top Down View** – In the future versions of MCC, the specified image will represent this map in the list of the maps in the menu of the game. However, this functionality is currently in the development and is not publicly available. The recommended file size of this image is around 900KB.
- **Max Teams All Up (Applies across the board)** – The maximum total amount of teams that can play on the map. If you set this value to "**4**", then the maximum amount of teams will be 4 regardless of the selected game type. See **Max Team Category (Applies per category)** below also.
- **Max Team Category (Applies per category)** – (*Optional, see the note below*) This section contains a lot of fields corresponding to particular multiplayer game types (**Slayer**, **CTF**, and so on). Every such field allows you to specify the maximum total amount of teams that can play on the map in this *particular game type*.

 > **NOTE**
 >
 > You should limit the maximum total amount of players either by the **Max Teams All Up...** field or by the **Max Team Category...** fields. If you specify both the **Max Teams All Up...** and **Max Team Category...** values, then the **Max Teams All Up** value will have priority and the maximum total amount of players will be defined by it regardless of the selected game type.

- **Default Primary Weapon** – This drop-down allows you to select the default primary weapon that will be given to all players on the map. The names of the values in this list (starting with the ***damage_reporting_type*** prefix) are related to the internal mechanics of the game and define how the

weapons are identified in the game and how they influence the statistics of the player. This drop-down list contains the full list of values, i.e. all values that are possible for this field in the game. Please note that these values are the *code names* (various e_damage_reporting_type values) and the specific *display names* can be different between each Halo title (e.g. Pistol vs Magnum). Values that can be used in practice (for particular Halo titles) are visually divided from unused values. (Actually, experienced modders may repurpose unused e_damage_reporting_type values in one of the Halo titles for their own new weapons.) And, even if you select one of the unused values here, nothing really bad will happen.

> **NOTE**
>
> **NOTE #1**: If you are creating a mod package for **Halo: CE**, then the **Default Primary Weapon** will be displayed not as a single drop-down list, but as a set of drop-down lists, allowing you to specify different default primary weapons for different modes of the game.

- **Default Primary Weapon in Forge** – This drop-down allows you to select the default primary weapon for all players on the map in Forge mode. See **Default Primary Weapon** above for details.
- **IMPORT MULTIPLAYER OBJECT TYPES** – This button allows you to import data from the **multiplayer_object_types.bin** file. This file contains an array of indices of tags that are used/available on the map. Different game variants require different resources (tags), so the data in this file affects the game variants (e.g. Slayer, Oddball, CTF, King Of Hill, Infection, and so on) that will be available for this multiplayer map. See Appendix B for details on the multiplayer_object_types.bin file and its generation using the modding tools.

> **IMPORTANT**
>
> This field is only relevant when modding Halo Reach, Halo 2 Anniversary, and Halo 4.

> **NOTE**
>
> **NOTE #1**: The name and the extension of this file must be exactly "**multiplayer_object_types.bin**" for the successful importing.
> **NOTE #2**: The **multiplayer_object_types.bin** file may be located outside the mod folder, since it won't be added to the mod package as the file – Excession will simply import values
> **NOTE #3**: If the **multiplayer_object_types.bin** file is empty or undefined for the map, only the Infection game variant will be available for the multiplayer map.

Firefight Map Properties

The **Firefight Map** panel allows you to specify the following properties of a Firefight map:

- **Allow Saved Films** – This option enables or disables the ability of players to save videos of their games on this map.
- **Image for Top Down View** – In the future versions of MCC, the specified image will represent this map in the list of the maps in the menu of the game. However, this functionality is currently in the development and is not publicly available. The recommended file size of this image is around 900KB.
- **Insertion Points** section – This section allows you to add information of insertion points of your Firefight map.

> **NOTE**
>
> **NOTE on Insertion Points**: In Halo, the player can start playing on the map not from the very beginning, but from the particular point of the mission (e.g. "Alpha", "Bravo", and so on). The Firefight mode of the game also allows you to start from different starting locations on the same map. Thus, insertion points of the map specify these locations where players start their game on the map.

> **NOTE**
>
> If your Firefight map is for **Halo 3: ODST**, you will need to set up an **Insertion Point** for Firefight specifically, you can also choose whether or not your Firefight map will record a saved film here. Halo: Reach, you do ***not*** need to set up **Insertion Points**.

Adding an insertion point is performed in a unified way, see Adding Insertion Points below for details.

After the properties of the insertion point are saved, it will appear in the **Insertion Points** section of the Firefight map.



Fig 2. View of the current insertion points within the map.

After that, if necessary, you can edit the properties of the added insertion point. To do that, select it in the list and click **EDIT**. Or, you can remove it from the map (select it and click **REMOVE**).

- **Default Primary Weapon** – This drop-down allows you to select the default primary weapon that will be given to all players on the map. The names of the values in this list (starting with the ***damage_reporting_type*** prefix) are related to the internal mechanics of the game and define how the weapons are identified in the game and how they influence the statistics of the player. This drop-down list contains the full list of values, i.e. all values that are possible for this field in the game. Please note that these values are the *code names* (various e_damage_reporting_type values) and the specific *display names* can be different between each Halo title (e.g. Pistol vs Magnum). Values that can be used in practice (for particular Halo titles) are visually divided from unused values. (Actually, experienced modders may repurpose unused e_damage_reporting_type values in one of the Halo titles for their own new weapons.) And, even if you select one of the unused values here, nothing really bad will happen.

Campaign Mission (Map) Properties

The Campaign Mission (Map) panel allows you to specify the following properties of a map within the campaign:

- **Allow Saved Films** – This option enables or disables the ability of players to save videos of their games on this map.
- **Insertion Points** section – This section allows you to add information of insertion points of your Campaign map.

> **NOTE**
>
> **NOTE on Insertion Points**: In Halo, the player can start playing on the map not from the very beginning, but from the particular point of the mission (e.g. "Alpha", "Bravo", and so on). The Firefight mode of the game also allows you to start from different starting locations on the same map. Thus, insertion points of the map specify these locations where players start their game on the map.

Adding an insertion point is performed in a unified way, see Adding Insertion Points below for details.

After the properties of the insertion point are saved, it will appear in the **Insertion Points** section of the Campaign map.



Fig 3. View of the current insertion points within the map.

After that, if necessary, you can edit the properties of the added insertion point. To do that, select it in the list and click **EDIT**. Or, you can remove it from the map (select it and click **REMOVE**).

Adding Insertion Points

To add an Insertion Point to the map properties:

1. In the **Campaign Mission (Map)** panel of a campaign map or in the **Firefight Map** panel of a Firefight map, in its **Insertion Points** section, click **ADD**.
2. In the appearing **Insertion Point** tab, specify the following properties of a new insertion point:
 - **Title** – The title of the insertion point.
 - Description** – The description of the insertion point.
 - Zone Set** – The identifier of the zone set of the insertion point (within the level itself). The format of this value depends on the target Title of the mod package:
 - For insertion points of **Halo 3** and **Halo 3: ODST** maps: this value is a number.
 - For insertion points of maps for ***all other Halo titles***: this value is a textual ID.

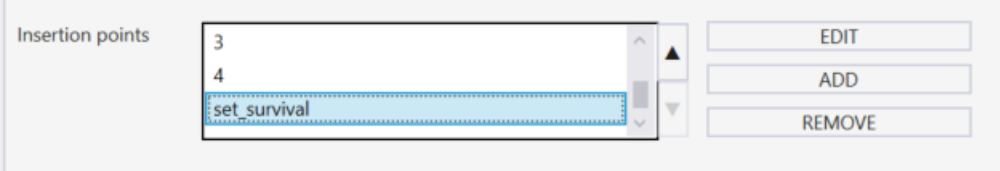
 > **NOTE**
 >
 > All resources of the level may be combined into different groups. The zone set defines the particular group of resources of the level. When the game switches between different parts of the level, the loading/unloading of the corresponding resources occurs. Thus, the zone set of the insertion point defines the resources that are necessary for loading to this location. Scenarios have a collection of zone sets. And every zone set has a corresponding id (or index) in this collection. For example, if you use a Halo 3 Editing Kit, you can view all zone sets of the level in Sapien – they are listed in the **Switch to zone set** drop-down (**Edit** > **Switch Zone Set** in the main menu, after opening of the level).

 - **ODST only** section – The properties if this section (listed below) are applied only in case of the ODST.
 - **Is Firefight** – If enabled means that this insertion point is related to Firefight mode.
 - **Return from Map GUID** – *(Optional field; not supported yet, see the Warning below)* This

value sets the GUID of the map, from which the players return to this insertion point. During the campaign progress, the game can switch players from one map to another (perform a "flashback"). When the player is returned from the flashback (returns back from another map), they can be returned to the particular insertion point. This is possible if the **Return from Map GUID** value of this insertion point is equal to the Map GUID of the map the player returns from.For example, if there are two maps: A and B, and the player is switched to B after A and, after that, should return from B to A (to the particular insertion point within A), then you will need to set **Return from Map GUID** of this insertion point to the Map GUID of map B.

IMPORTANT

If your Firefight Map doubles as a Campaign mission, you will need to add any **Campaign Insertion Points** that are indexed *before* the **Firefight Insertion Points** or else your map will fail to load properly.



WARNING

Currently, the support of this field by Halo titles is in the development and this field will not work for them. The support of this field will be added in the next versions of MCC.

3. To save your changes made in the **Insertion Point** tab, select **File** > **Save** or **File** > **Save All** from the main menu of Excession.

Viewing and Editing a Mod Package

11/23/2022 • 2 minutes to read

After you have finished the creation of your mod package and have saved all its contents, you can click the **REFRESH** button in the **Home Screen** tab of Excession.

This will refresh the list of mods visible to the tool and, if your mod package is correct at least partially, it will be displayed there.

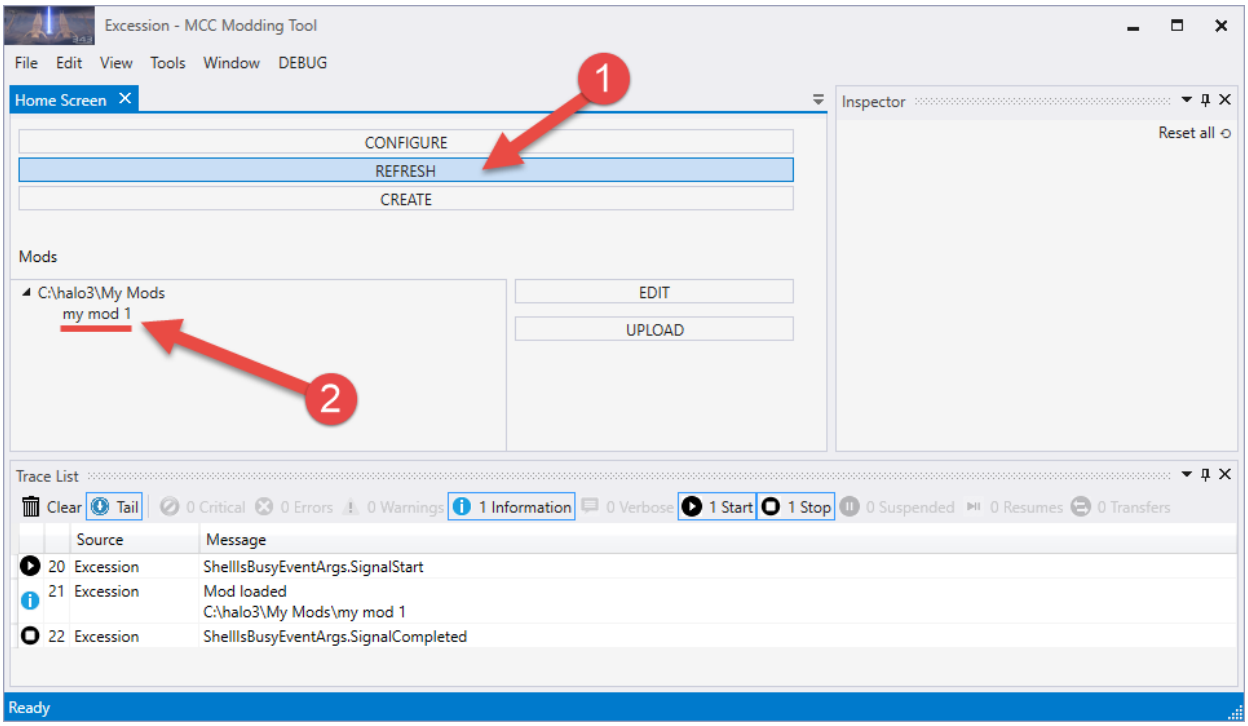


Fig 1. View of the Home screen of the Exession Tool, highlighting the Refresh button and the list of your mods.

Along with that, you can check the list of errors, warnings, and notification messages displayed in the log within the **Trace List** panel. If your mod was not found by Excession or has any issues, there will be errors or warnings there.

To view the detailed information about a log entry, select it within the **Trace List** panel and view the information displayed in the **Inspector** panel.

To edit a created mod, select it in the **Mods** list, and click **EDIT** next to the list of mods. The hierarchy of tabs corresponding to various components of a mod package and their UI during editing is absolutely the same as during their creation.

Uploading a Mod Package

11/23/2022 • 2 minutes to read

> **WARNING**
> To be able to upload a mod to Steam Workshop, you need to be logged into your Steam client and have Steam client up and running. If you are not logged into it or it is not launched, you will not be able to upload mods via Excession – the **UPLOAD** button will be dimmed in this case and you will see a corresponding message in the UI of Excession.

After you have finished the creation of your mod package, you can upload it to Steam Workshop.

To do this:

1. In the **Home Screen** tab, select the necessary mod package in the **Mods** list.
2. Click **UPLOAD** next to the list of mods.

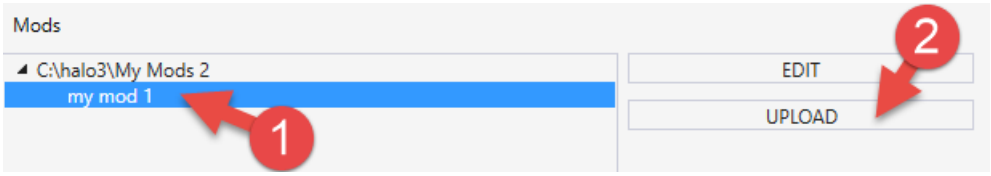


Fig 1. View of the your selected mod and the Upload button.

3. In the appearing **Upload** tab, do the following:
 - Ensure that the **Title** and **Description** of the mod package are correct. If you need to edit them, stop the uploading process and correct them while editing the mod package.
 - If necessary, in the **Choose Image for Mod...** field, select the image that will represent your mod in Steam Workshop. (This image is optional).

 > **NOTE**
 > This image must be in a PNG or JPG format. Its size must be not greater than 1MB. However, this image can be stored outside the mod folder, since it will not be included into contents of the mod package and will be uploaded separately.

 - Select the "**I have read and accept the terms of the Steam Workshop Contribution Agreement**" option once you have read the documents. You can view the FAQ and the Agreement by clicking the corresponding buttons below this option.
4. Click **UPLOAD**. (If it is dimmed, launch Steam client and log into it, see **WARNING** above.)
5. After that, Excession starts uploading your mod package to Stream Workshop. After the uploading is finished, you will be able to find it in Steam Workshop. For example, you can locate it within the list of your mods at **[Your Profile] > Workshop Items > Halo: The Master Chief Collection**.

Appendix A: UI of Excession

11/23/2022 • 2 minutes to read

After the initial configuration and creation of some mods launch (see above), the UI of Excession will look similarly to the following:

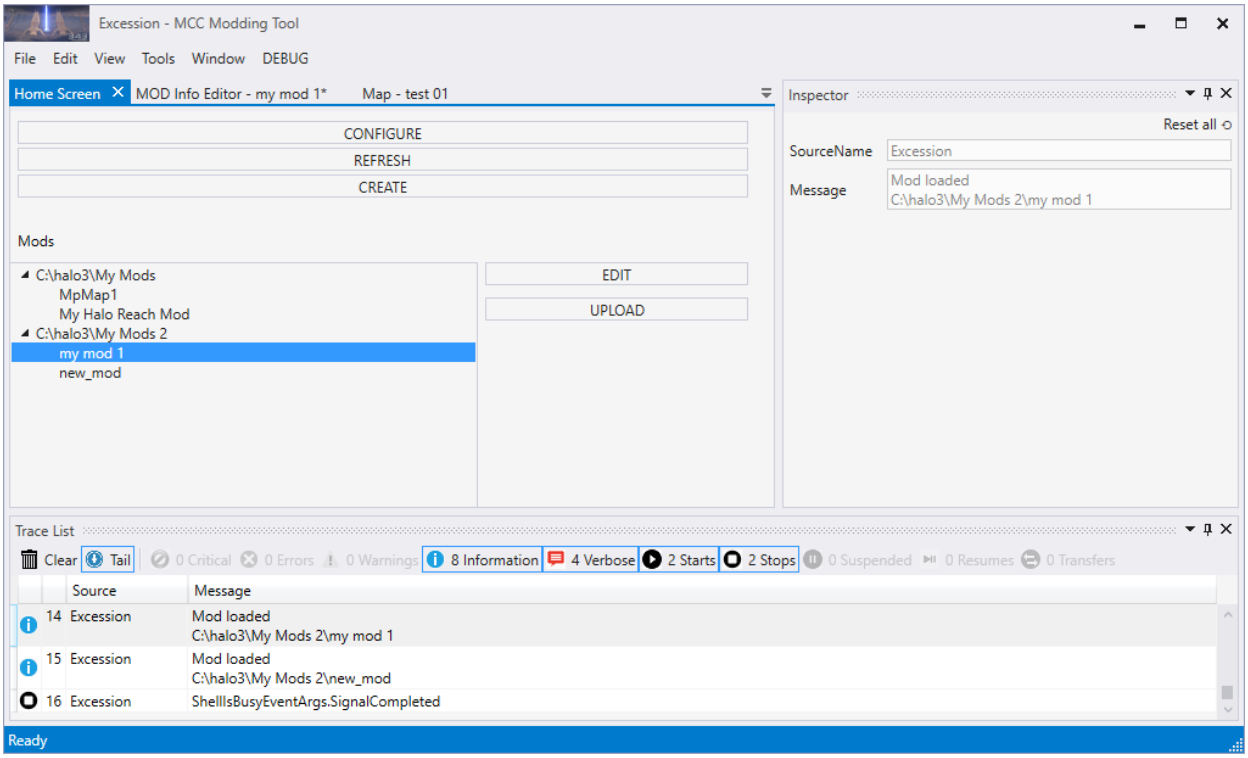


Fig 1. The UI of Excession just after initial configuration and creation of some mods.

The main window of Excession displays tab that are necessary for working with mods. The main tab is a **Home Screen** that displays all your mods with the configured root folders in the **Mods** list, allows you to create new mods (**CREATE**), change existing mods (selection in the list + **EDIT**), and upload them to Steam Workshop (selection in the list + **UPLOAD**). And, also allows you to configure your root folders (**CONFIGURE**) and refresh the list of mods (**REFRESH**).

In the process creating/editing a mod, the tool will open new tabs for you that will allow you to edit properties of various entities contained by the mod (of the mod itself, of a particular map, episode, insertion point, and so on). To save your changes to these properties, select **File** > **Save** or **File** > **Save All** from the main menu of Excession after your modifications.

> **NOTE**
>
> Editing of field values in these tabs has one specificity: when you change the value of the field, you need to remove the focus from this field before saving, otherwise the new value will not be saved.

The **Trace List** panel (at the bottom of the main window) displays the log of all operations performed in and by Excession. For example, new entries are added to it when you perform **REFRESH** to identify mod packages in the root folders.

If necessary you can filter this log by debug levels (by **Critical**, **Errors**, **Warnings**, and other similar filters), enable or disable auto-focus on the most recent log entries (the **Tail** switcher), and so on.

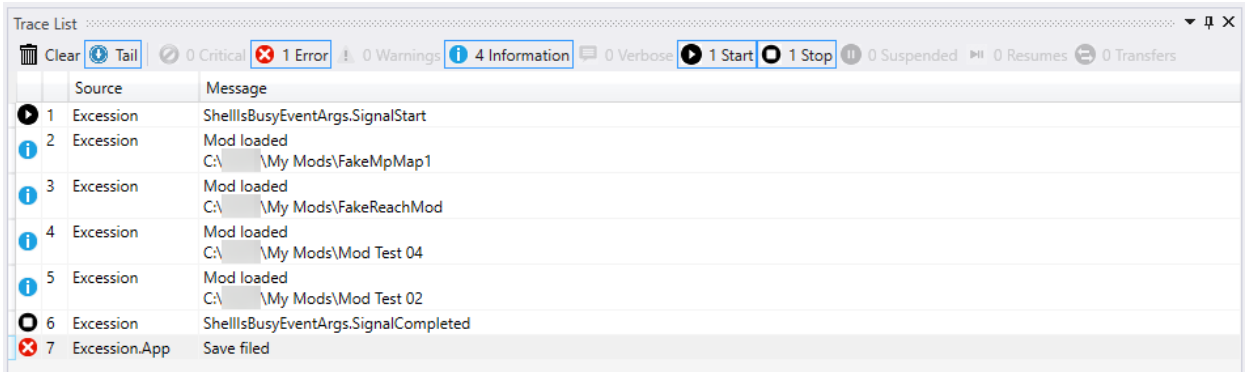


Fig 2. The Trace List filtering by log types.

The **Inspector** panel (at the right side of the main window) displays detail for log entries selected in the **Trace List** panel. This is useful for viewing details of errors.

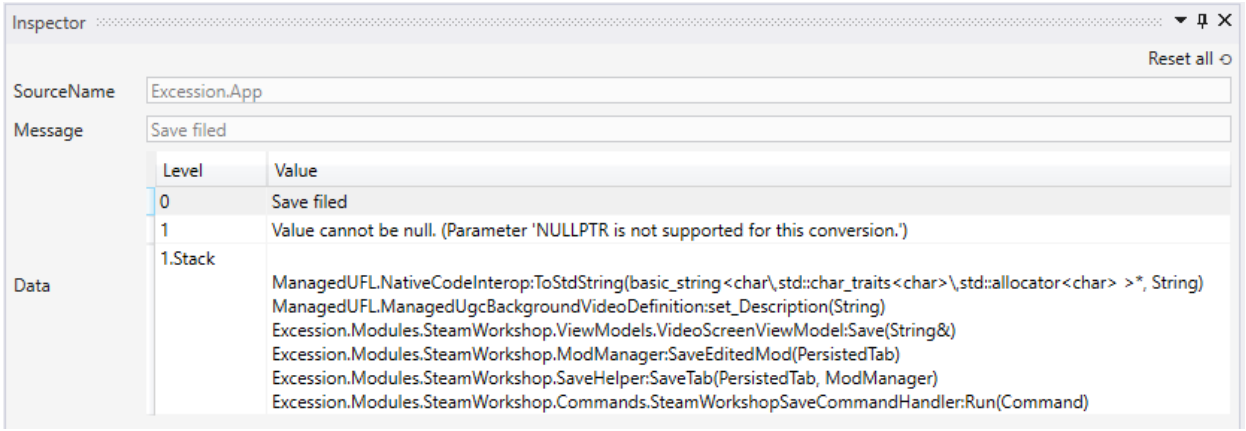


Fig 3. The Inspector panel showing more details of an error.

Appendix B: How to generate the "multiplayer_object_types.bin" file?

11/23/2022 • 2 minutes to read

The **multiplayer_object_types.bin** file allows the system to define the game variants (e.g. Slayer, Oddball, CTF, King Of Hill, Infection, and so on) that will be playable on the particular multiplayer map.

> **NOTE**
>
> **NOTE #1**: You are able to import data from the **multiplayer_object_types.bin** file to a multiplayer map when defining its properties in Excession, using the **IMPORT MULTIPLAYER OBJECT TYPES** button – see Multiplayer Map properties for details.

> **NOTE**
>
> **NOTE #2**: If the **multiplayer_object_types.bin** file is empty or undefined for the map different behavior will occur depending on the title.
>
> For Reach, only the Infection game variant will be available for the multiplayer map.
> For Halo 2 Anniversary, maps will **not** appear at all in the Custom Game lobby.
> For Halo 4, it will show Slayer, Flood, and Regicide as available game variants.

This file contains the list of entities that are provided by the map (e.g. its weapons, vehicles, grenades, and so on). The contents of this list define the compatibility of this multiplayer map with certain game variants.

The **multiplayer_object_types.bin** file can be generated for the map by one of Halo Modding Tools – **tool.exe**. This applies to Halo Reach, Halo 2 Anniversary, and Halo 4.

Alternatively, while a scenario is loaded in a **Standalone (Tag Test) tool**, running this command "**motl_dump_bitvector_for_scenario**" will generate the multiplayer_object_types.bin file in the root of the title's tools folder.

To generate this file you need to use the **multiplayer-generate-scenario-object-type-bitvector** command in this tool. The syntax of this command is the following:

```
multiplayer-generate-scenario-object-type-bitvector <scenario> <cache_file_resource_gestalt> <output_file>
```

Where:

- **<scenario>** – The path to the **.scenario** tag file of the map. This path should be specified relative to the **tags** folder of your Halo Editing Kit directory and should include the name of the **.scenario** tag file *without* the file extension.
- **<cache_file_resource_gestalt>** – The path to the **.cache_file_resource_gestalt** tag file, which is generated during the process of final building of the map (creation of the .map file). This path should be specified relative to the **tags** folder of your Halo Editing Kit directory and should include the name of the **.cache_file_resource_gestalt** tag file *without* the file extension.

 The file is located in the [Root Mod Tools Folder]\reports\[map].

 - For example: Deadhead\reports\52_ivory_tower\52_ivory_tower.cache_file_resource_gestalt

 That file should be copied into the tags folder. It is recommended to use the map's scenario folder:

 - For example: Deadhead\tags\levels\multi\52_ivory_tower
- **<output_file>** – The path to the resulting **multiplayer_object_types.bin** file.

For example:

```
tool.exe multiplayer-generate-scenario-object-type-bitvector path/to/scenario/name path/to/gestalt/name
multiplayer_object_types.bin
```

Halo CE Mod Documentation Coming Soon

11/23/2022 • 2 minutes to read

Halo 2 How-Tos Home

11/23/2022 • 2 minutes to read

There are various guides available for Halo 2 which contain additional information you may need while creating mods.

Build Cache (MAP) File

A cache file is a compilation of all of the tags associated with a scenario, condensed into a single file called a .map tag.

Create Effective Bitmaps

Information on how to effectively compress your bitmaps for the best performance and visuals.

Dynamic Lights Best Practices

Information on the best practices when using dynamic lights.

Fix "Creeping" Idle Animations

Information on how to fix "creeping" idles, which is where characters may slowly "slide" or "creep" from their starting location.

Information on material_effects

General guidelines for setting up material effects in objects and the divisions between the art and sound pipelines.

Set Instanced Geometry Pathfinding Policy

Information on setting Pathfinding Policies for Instanced Geometry.

Set Object Reset Heights in MP

Information on the height at which netgame objects like flags and balls reset to their original location is set globally.

Seting Up A Character with Basic AI

Information on how to set up a character with basic AI.

Set Up Controller Cheats

Information on how to set up a cheats.txt file that allows you to invoke console commands from the game controller.

Set Up King of the Hill in MP

Information on how to set up King of the Hill in your multiplayer scenario.

Set Up Teleporters in MP

Information on setting up teleporters in your multiplayer scenario.

Split Scenario Resources

Information on how a .scenario file can be split up into components so that several people can work on different sections of the game at the same time.

Using screenshot_cubemap Command

Information on a console command that renders a cube map of an environment from the viewpoint of the camera.

How-To - Build a Cache (Map) File

11/23/2022 • 2 minutes to read

A **cache** file is a compilation of all of the tags associated with a scenario, condensed into a single file called a **.map* tag. The resulting **.map** tag resides in the H2EK\h2_maps_win64_dx11\ folder.

Running the game from a cache file is a good approximation of what the final build will be like. Tag builds allow you to quickly modify and sync assets for the game, but the only way to test what the shipping fill rate will be is to run a **cache build**.

BUILDING MAP WITH TOOL

You can build a cache file (the .map tag) using the tool command **build-cache-file**. This will build the .map tag from the scenario tag that you specify.

```
build-cache-file [scenario]
```

For example, the command to build and run the cache file for Alpha Gas Giant would be:

```
tool build-cache-file scenarios\solo\alphagasgiant\alphagasgiant
```

More information on how to set up a mod can be found in the Excession Tool documentation.

BUILDING MAP WITH GUERILLA

1. Choose **File → Run Tool**....
2. Select tool command **build-cache-file**
3. Select the .scenario file that you want to build the cache file for.

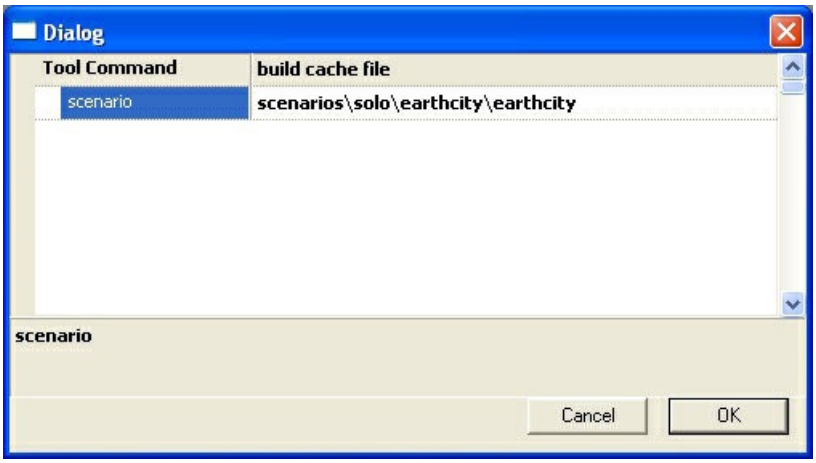


Fig 1. build-cache-file with the earthcity scenario selected.

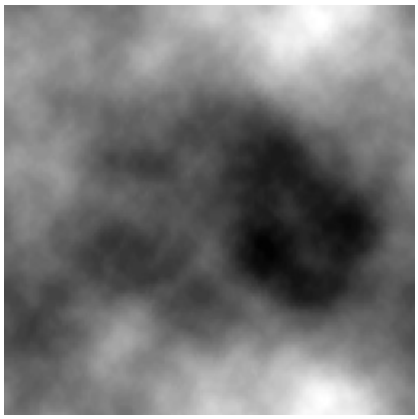
How-To - Create Effective Bitmaps

11/23/2022 • 2 minutes to read

Everyone understands that compression affects the quality of bitmaps but it's important to also understand that there are ways to make the most of the bits we have.

Bitmap Values

The most important thing is to use the full value range whenever possible. There is a maximum of 256 values in any bitmap - no matter what the compression scheme. Authoring bitmaps that use a limited value range will create posterized bitmaps.

USE THIS	NOT THIS
	

It's possible to "turn down" a bitmaps value range in the game. For example in the specular (which uses the alpha channel of the base_map) you can use a darker color for the specular_color, which will allow you to use a wider value range that gives you better control over the effect you are trying to achieve.

Here's an example with a cloud map. The left half of this image uses a full value range bitmap (0-255) that is scaled in the tag while the right half uses a bitmap with a value range of 0-63:

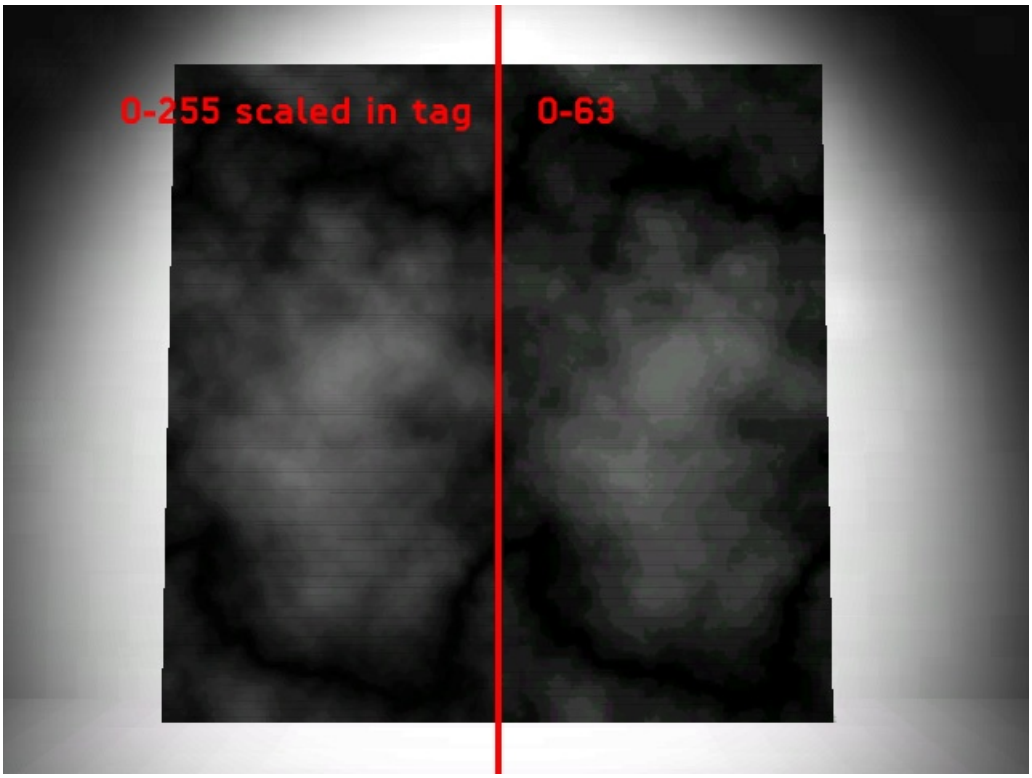


Fig 1. Difference between full range of 0-255 on the left and a range of 0-63 on the right.

This actually makes a bigger improvement than increasing the bit-depth. In this final example, the left half of this image uses a monochrome compression setting (8 bits/pixel) while the right half used DXT1 (4 bits/pixel). The left half (monochrome) uses twice as much memory as the right.

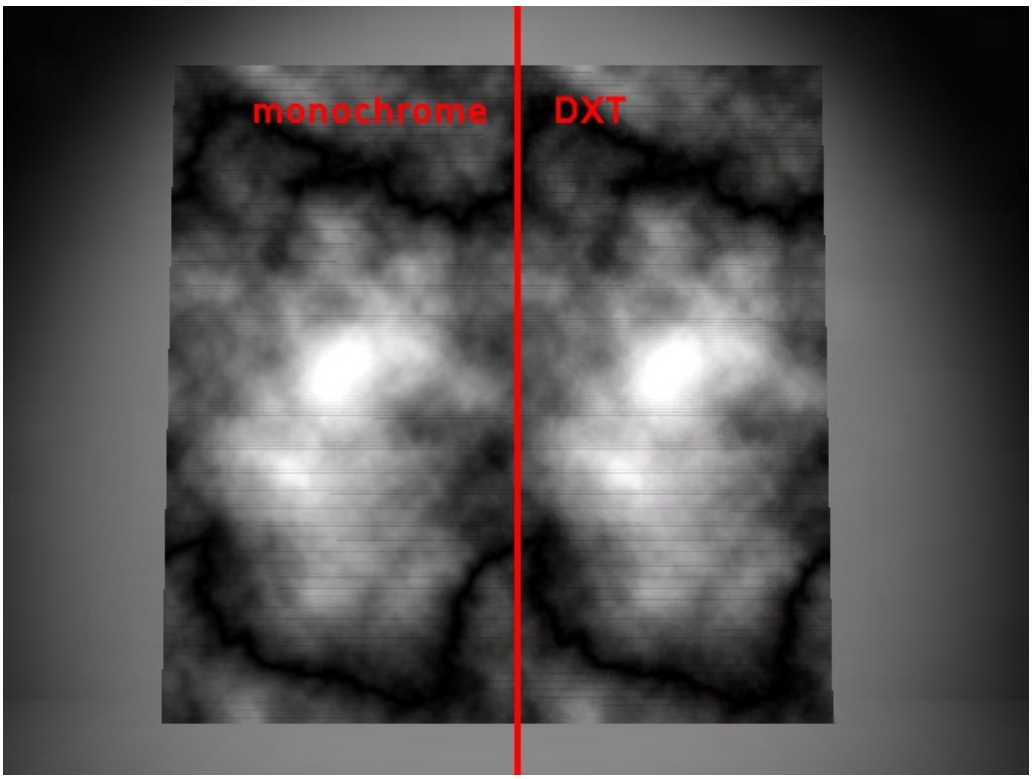


Fig 1. Difference between monochrome and DXT1 compressions.

How-To - Dynamic Lights Best Practices

11/23/2022 • 2 minutes to read

Dynamic lights are fairly expensive in general. You should never have more than 4 reasonably sized lights in view at any given time. A reasonable size is less than 10 world units square at the base of what it's shining on. The Halo 2 shadowing techniques will not look good for any light larger than this. Because they are so expensive, these are some guidelines to make them cheaper:

- Use the smallest FOV on the light you can. This makes your shadow look better and gives a tighter fitting bounding sphere.
- Tesselate the environment in the area where the light is hitting so that we're not re-rendering the entire floorway of your room for each one. Dynamic lights are in general predominantly fillrate bound.
- Set a closer shadow or illumination fade distance so both systems can deactivate
- Use these flags whenever you can:
 - doesn't shadow environment (if you don't obviously see shadow off of structure, disable them!)
 - doesn't shadow parent (shadows not cast from object you are attached to)
 - ignore parent object (neither shadows nor illumination on/from object you are attached to)
 - no stencil shadow (no shadows at all on this light)
 - doesn't shadow objects (do not cast shadows from objects in the light)
 - no specular (especially on sphere lights, it's a killer there)
 - set the shadow antialiasing taps to 1-tap
- set the shadow quality bias (buffer resolution) negative (turn on rasterizer_shadow_buffer_debug and it will show you the actual resolution currently being used by the shadow buffer)
- pull the cutoff plane in as close to behind what you are shining on as possible so we get the best sphere fit (turn on debug_lights to see the sphere fitting - this sphere is also where we lod and fade the light from)
- don't set a very far distance between the falloff and cutoff plane. In general we don't need a huge fade interpolation distance built into lights.
- Be careful placing large lights and be thoughtful of which portals it will be shining through (it may be lighting clusters you don't realize)

Use Debug Commands

debug_lights - shows you the light geometry, the bounding sphere used for lod'ing, shows you if it is using specular (2 passes means specular, 1 pass is just diffuse). Also you can see all active lights.

rasterizer_shadow_buffer_debug - shows you the resolution of the shadow buffer being used (only displays if shadows are active)

render_layer_enable diffuse or specular - 0 or 1 to see how much the light's diffuse or specular are effecting the scene

How-To - Fix "Creeping" Idle Animations

11/23/2022 • 2 minutes to read

If your custom character has an incorrectly set up idle animation, they may slowly "slide" or "creep" from their starting locations.

The most likely source of this error is where idle animations are .JMA files that contain pelvic-animation.

All .JMA files have their pelvis motions translated into dx,dy data which is used by the physics system to move the actual unit per frame. This movement is not exact, and will cause the character to creep across the map in looping animations like the idle.

To prevent this, all Idle animations should be .JMM files, just like enter and exit animations. This will prevent the physics stuff from happening and keep your character locked in place while animating. This not only fixes the creeping problem, but also helps to lock the feet to the ground much better.

> **NOTE**
> This should not occur often, as Tool treat all JMAs with the word 'idle' in them as if they were JMM files.

How-To - Information on material_effects

11/23/2022 • 2 minutes to read

General guidelines for setting up material effects in objects and the divisions between the art and sound pipelines.

OVERVIEW

There are many types of objects which have a material_effects slot in them (crates, vehicles, bipeds, etc.). Generally, you should leave this slot empty and allow the surface types of the various objects to generate art effects and sound fx as it is laid out in globals.globals.

There are several instances where we you will hook up a material_effects tag to that slot in a specific object:

- art or sound wants to treat the object as a single entity instead of the collection of various pieces (think warthog). here are two objects of the same surface type where the sound or art just does not fit for both... we would then special case one of them and let the other one be taken care of by globals
- all bipeds will have material_effects tags assigned to them for footsteps. However as long as the only events in these tags are footstep events all other aspects of havok interaction (collisions, grinding, rolling) will be taken care of by globals according to surface type.
- As a general rule, avoid using material_effects tags hooked up at the object level. It adds complexity to an already complex hierarchy. However given that art and sound may want to special case something at the object level we need to follow

A SIMPLE RULE

If, for instance, an audio guy decides to special case sound fx for an object they make a new material_effects tag and hook it up at the object level. This tag should not have any ART elements present in the ART tag block area. So if perchance that audio guy started with another material_effects tag and did a "save as" to a new tag that very same audio guy should delete all of the art references in the art tag block area. This will allow sound to override at the object level, but art effects to proceed to globals and resolve surface type effects there. And vice versa.

In a nutshell if anyone hooks up a material_effects tag at the object level it should only have references in it for their own discipline.

The following are template material_effects tags which can be used as a starting point and will have all references from the "other side" already removed.

```
effects/materials/sound_template.material_effects

effects/materials/effects_template.material_effects
```

Also as a reminder ALL material_effects tags should live somewhere in the directory structure under tags\effects\materials...

How-To - Set Instanced Geometry Pathfinding Policy

11/23/2022 • 2 minutes to read

Instanced geometry ("Poops") can have a pathfinding policy specified in their name by adding a special character just after the '**%**'.

```
%mylittlepoop01
```

Poops **without any special characters** default to pathfinding policy "**cut-out**". This means that ai will pathfind around them but will not be able to walk on them. This is good for things like columns which are z-buffered into the floor. Of the three poop pathfinding policies, this is in the middle in terms of memory usage.

```
%-mylittlepoop02
```

Poops with a **minus** just after the percent will use pathfinding policy "**none**". This means that ai will pathfind as if the poop wasn't there. This is good for things that aren't near playable space like rafters on the ceiling. This uses the least amount of memory.

```
%+mylittlepoop03
```

Poops with a **plus** just after the percent will use pathfinding policy "**static**". This means that ai will fully pathfind on them. This is good for things like catwalks, stairs, etc. It uses the most amount of memory.

These pathfinding policies are the same as the ones we can specify for scenery and machines, but they have one more called "dynamic".

What about pathfinding policy "dynamic." Since poops can't move, they don't need the dynamic pathfinding policy and it's much more efficient to bake it into the pathfinding graph.

How-To - Set Object Reset Heights in MP

11/23/2022 • 2 minutes to read

The height at which netgame objects like flags and balls reset to their original location is set globally. You'll want to reset this to a more reasonable value for your level so that objects don't fall forever before resetting. This can be done in the Scenario tag, about 2/3 of the way down.

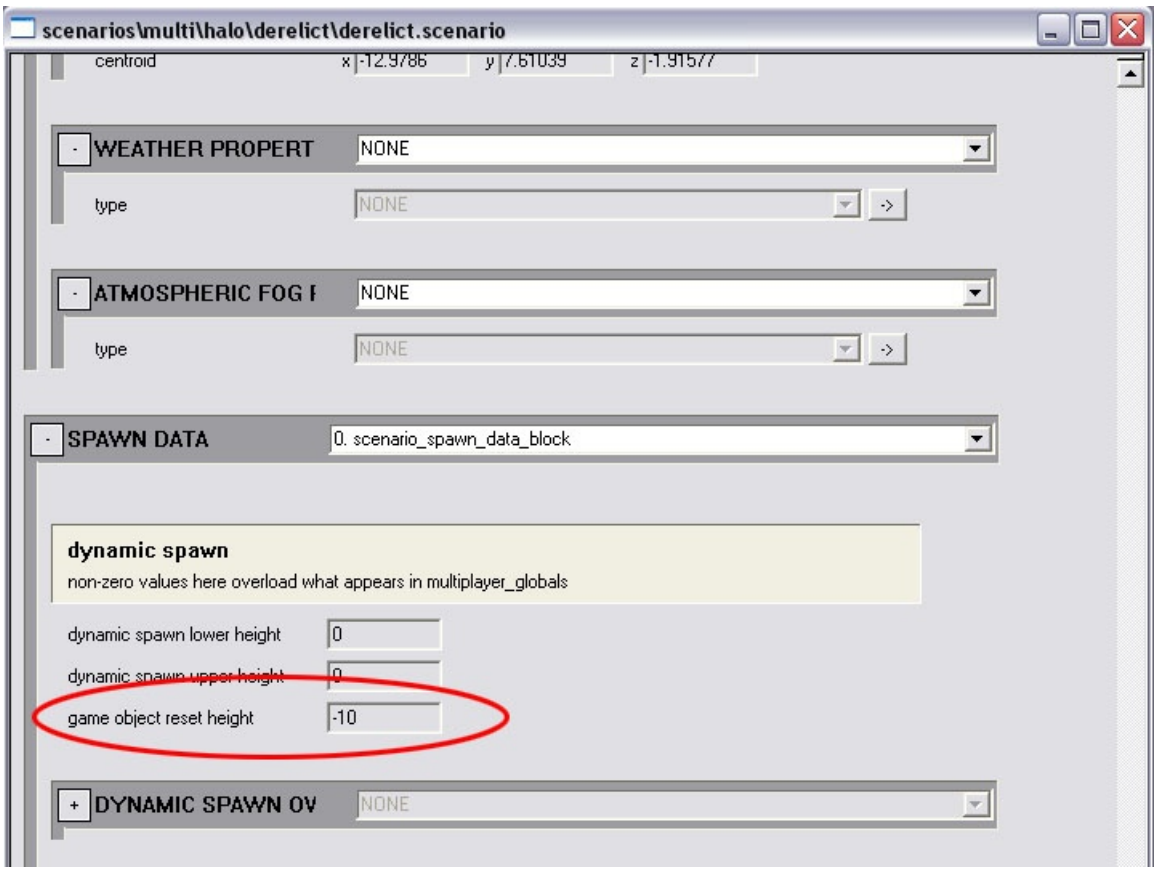


Fig 1. The game object reset height field in the scenario tag.

How-To - Setting Up A Character With Basic AI

11/23/2022 • 2 minutes to read

BASIC SETUP

character tag must be set up first

object\character\[character]\ai designers should set this up

OPEN SAPIEN

- add character
- add weapon
- add tags\ai\style (usually: guarding)
- Go -> Everything\Mission\AI\Zone
 - # zone = collection of firing positions
- New Instance # creates new zone
 - expand new zone
- Go -> Firing positions
 - drop several (~6) firing positions by r-click on scenario in Game window # firing positions must belong to area
- Go -> Area
- new instance # creates new area
 - assign firing points to new area
- Go -> Firing positions
 - select firing points # in Game window or Hierarchy view
 - in Properties palette
 - area: assign to area # ignore NO sign next to FPs in Hierarchy view
 - can select Firing points and Ctrl-N creates new area and assigns FP to new area
- Go -> Squad
 - new instance # creates new squad
 - in Properties palette
 - set name # this will be what is called from console
 - set normal diff count = same as insane diff count count (~2) # of characters to spawn
 - set actor defaults
 - # ignore vehicle
 - character type # to be spawned

 - initial zone
 - initial weapon
 - (optional: grenade type)
 - # ignore initial order
- Go -> Squad
 - expand specific squad
- Go -> Starting points
 - place starting points # [normal diff count] will spawn for each

IN CONSOLE

console: ai_place [squad_name]

How-To - Set Up Controller Cheats

11/23/2022 • 2 minutes to read

Any console command can be set in your **init.txt** file. Similarly you can set up a **cheats.txt** file that allows you to invoke console commands from the game controller.

> **NOTE**
> Create cheats.txt in your H2EK root directory.

In your **init.txt**, add a line of text that says **cheat_controller on**

To activate your new commands, press the BACK button and then the corresponding button to the command you wish to activate.

> **NOTE**
> THIS WILL ONLY WORK WHILE YOU ARE IN CONTROL OF THE CHARACTER. you won't be able to invoke commands while in the flying camera mode.

FORMAT

Format for boolean commands (ones that accept true,false or on,off or 1,0) is:

```
set [command] (not [command])
```

- example: set debug_objects (not debug_objects)

Format for cheat commands is simply:

```
[command]
```

- example: cheat_teleport_to_camera

Format for integer commands is a bit trickier. These are commands that require a numerical value (like pad3):

```
set [command] (- 1 [command])
```

- example: set pad3 (- 1 pad3)

Format for unmapped keys:

```
empty (empty)
```

ORDER OF COMMANDS

The order of commands is assumed - ie, the first line is always mapped to the "A" button, the second line to the "B," etc.

The order of commands is:

- ine 01 = A button
- line 02 = B button
- line 03 = X button
- line 04 = Y button
- line 05 = Black button
- line 06 = White button
- line 07 = Left Trigger
- line 08 = Right Trigger
- line 09 = D-Pad Up
- line 10 = D-Pad Down
- line 11 = D-Pad Left
- line 12 = D-Pad Right

ADDING COMMENTS

You can add comments to any line by seperating it with a semicolon.

example: set debug_objects (not debug_objects); A button

The words "A button" don't actually do anything - the code stops reading the line at the semicolon (;). It's just a good way to show someone looking at the file what this line of code is doing.

SAMPLE CHEATS.TXT FILE

- set debug_objects (not debug_objects); A button
- set ssc_shadow (not ssc_shadow); B button
- set render_model_markers (not render_model_markers); X button
- set render_model_vertex_counts (not render_model_vertex_counts); Y button
- set debug_render_freeze (not debug_render_freeze); Black button
- cheat_teleport_to_camera; White button
- set ssc_bump_mode_debug (not ssc_bump_mode_debug); Left Trigger
- set rasterizer_bump_mapping (not rasterizer_bump_mapping); Right Trigger
- empty (empty); D-Pad Up
- empty (empty); D-Pad Down
- empty (empty); D-Pad Left
- empty (empty); D-Pad Right

How-To - Set Up King of the Hill in MP

11/23/2022 • 2 minutes to read

Hills are differentiated by their team designator. The team marked "Alpha" team is the one used in a non-moving hill game. Hills with an identifier of "0" are used to mark the hill boundary.

Each boundary point in a hill is one of two handles defining a point on a bezier curve--the hill periphery. I know, it's ugly. The first point placed is the second handle of the first point, the next is the first handle of the next point, and so on, until you close the loop with your last netgame flag being the first handle of the first point. **Your total number of points defining the perimeter must be even!**

Points with an identifier of "1" are not used to define the perimeter, but instead are used to define the upper and lower boundaries of the hill. These can be left off and the hill will use default heights, but the bottom will be level with the perimeter points, resulting in a hill where even the smallest depression causes a player to leave the boundary. Typical hills have an even number of boundary points, a point marking the lower bounds, and a point marking the upper bounds.

How-To - Set Up Teleporters in MP

11/23/2022 • 2 minutes to read

Teleporters

Teleporter source and destination locations are indicated with teleporter netgame flags. These are placed in Sapien, in the "mission\game data\netgame flags" directory.

Each teleporter requires a separate pair of source (src) and destination (dest) flags. A pair is defined by a matching identifier number. The team designator is not used for netgame flags, and so should be set to "Neutral" to avoid confusion.

Direction on a teleporter is tricky business. The facing on the source netgame flag should match the predicted direction of entry, the facing on the exit flag should match the desired direction of facing on exit. Sometimes you have to tweak this to get it right.

A teleporter flag has no associated model, that has to be placed independently as scenery. Generic teleporter scenery exists in "tags\objects\multi". Teleporters can also be designed into the environment, like Derelict from Halo 1 or Relic from Halo 2.

How-To - Split Scenario Resources

11/23/2022 • 2 minutes to read

A .scenario file can be split up into components so that several people can work on different sections of the game at the same time. For example. with a split mission one person could be working on placing characters while another was working on placing scenery.

This article shows you how to split up a scenario file into 11 scenario resource files.

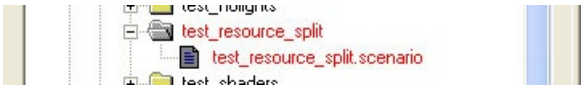


Fig 1. A single scenario resource file.

1. Open the scenario in Sapien.
2. Under the File menu you'll find a command for Split mission resources. Click on that to split the scenario.

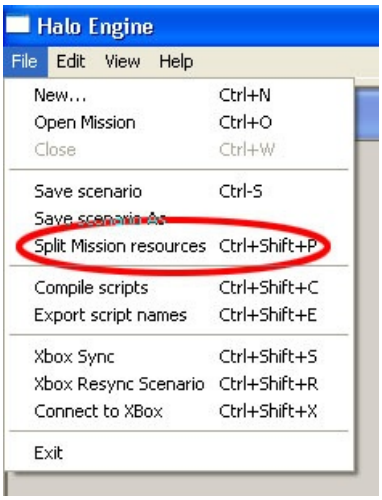


Fig 2. File > Split Mission Resources.

After a second you should see a message indicating success in the lower left corner.

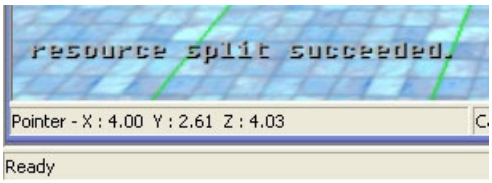


Fig 3. Resource Split Succeeded message.

In the directory that contains the scenario you'll see a new folder called resources. This contains all of the new resource files.

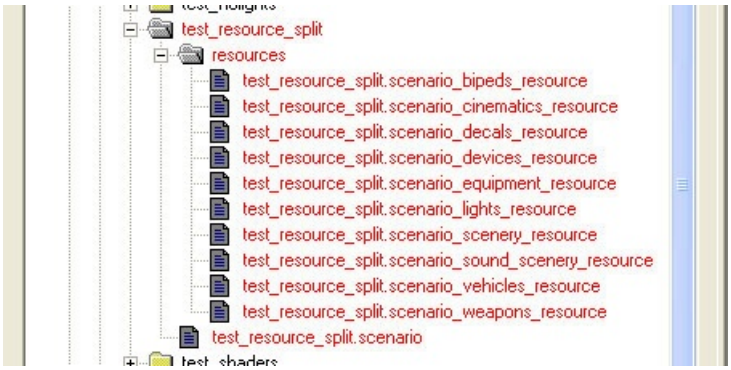


Fig 4. Newly created resource files.

MAKE SURE YOU ADD THE RESOURCE FILES TO THE DATABASE - don't check in the converted .scenario file without adding the new resource files, too.

> **WARNING**
> Once a file has been split it cannot be reassembled!

scenario_biped_resource

palette and instances of bipeds

scenario_cinematics_resource

palette and instances of flags, camera points and recorded animations

scenario_decals_resource

palette and instances of decals

scenario_devices_resource

palette and instances of all device groups, machines, controls and light fixtures.

scenario_equipment_resource

palette and instances of equipment

scenario_lights_resource

palette and instances of lights (does not include light fixtures).

scenario_scenery_resource

palette and instances of scenery

scenario_sound_scenery_resource

palette and instances of sound scenery

scenario_vehicles_resource

palette and instances of vehicles

scenario_weapons_resource

palette and instances of weapons

How-To - Using screenshot_cubemap Command

11/23/2022 • 2 minutes to read

There is a console command called

```
screenshot_cubemap
```

...that renders a cube map of an environment from the viewpoint of the camera.

How to Use

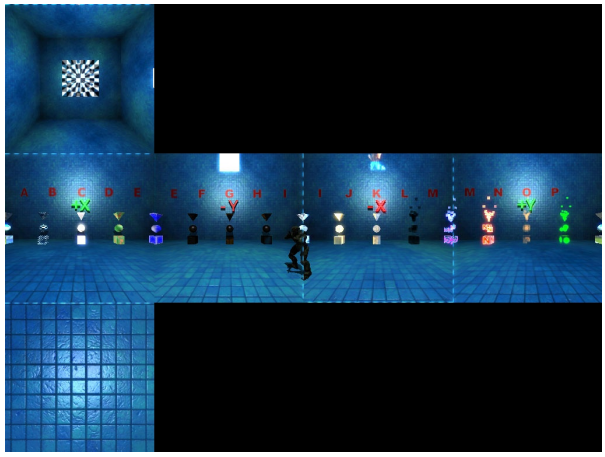
Position the camera at the point where you want to render the cubemap. It doesn't matter which way the camera is pointing - the cube map is always rendered in the correct XYZ orientation.

> **NOTE**
> that you must be in the flying camera, otherwise you will render the HUD into the cube map.

Open the console (by pressing the ~ key) and type in:

```
screenshot_cubemap [name]
```

This will create a TIF file called name.tif in your Halo 2 directory. You can then convert it into a bitmap tag for use in the game as an environment map.

SCREENSHOT LOCATION	GENERATED CUBEMAP
	

AI Overview

11/23/2022 • 6 minutes to read

BEHAVIORS

- **Formations**: Any homogenous collection of characters can group itself into a formation. However, there are currently no rules dictating when this should be done, since it is largely assumed that most formations will be scripted. This is an appropriate way, for example, to do AI insertions. (e.g. player enters a clearing just in time to see a jackal phalanx march around the corner). A formation behavior could, for example, be initiated by the level designer in an HS command script that aborts when the AIs become aware of an enemy. Thus the formation holds until the player opens fire, and then everyone scatters.

MOVEMENT

Environmental Pathfinding Hints

- **Jump hints**: Jumping connections can be designated between unconnected pieces of geometry. The AIs will then be able to pathfind through these hints. Note that jump-hints take the form of two parallel rails, where a point on the departure rail takes the AI to the corresponding point on the arrival rail. (this is as opposed to a single point-to-point connection).

- **Climb hints**: Climbing hints can be placed on certain configurations of geometry. This requires two levels of floor connected through a perfectly vertical wall. Unlike the jump hints, climb hints are point-to-point, like a ladder.

- **Pathfinding on moveable objects**: There are two types of objects that can be set to be pathfindable, i.e. that AI will be able to navigate on: scenery and machinery. While scenery is static (and therefore look, to pathfinding, like just another piece of the environment) machinery can animate in arbitrary ways. Examples of this include elevators, and Marcus's magic floating platforms. Firing positions can be placed on these machines, and the AI will behave and move just as they would on the ground. In order to connect these machines to the rest of the environment, **jump hints** must be placed between suitable entry/exit points.

 In the future, moveable pieces of bsp will also be navigatable in this way.

Object Hints

In addition to hints that are explicitly placed in the environment by the level-designer, special hint-markers can be added to object render_model definitions which will create hints automatically for every placed instance of that object.

- **Jump hints**: The markers must be names hint_jump_Xa, hint_jump_Xb, hint_jump_Xc and hint_jump_Xd, where X is the number of the hint, in two-digit format starting from 01 (i.e. 01, 02, 03, ... 10, 11, 12 ...) Each set of four markers defines, as in the explicitly-placed environment hints, a set of two rails, with a and b defining the departure rail and c and d defining the destination rail.

> **NOTE**
>
> that the geometric configuration of the hint markers must be the same as the environmental ones, with a and c being on the same side and b and d being on the same (stated another way, if you took a vector a→b and another vector c→d, both vectors would be pointing in roughly the same direction).

- **Climb hints**: Only two markers are required, name in the format hint_climb_Xa and hint_climb_Xb. Like their explicitly-placed counterparts, the order must be from the lower surface (a) to the upper surface (b).

POSTURES

- **Wall leaning**: In certain circumstances, AIs will assume special covering postures near a corner providing cover (e.g., marines leaning their back against the wall). In order for this to happen there are two requirements: (a) a firing position needs to be placed near the corner (within about 0.4 wus). (b) the geometry must consist of floor connected directly with vertical wall (such that, for example, there can be no bevel at the base of the wall). The wall must also be of sufficient height (about 0.6 wus) and breadth to hide a human-sized character.

OBJECTS

Pathfinding with Obstacles

Usually AIs just pathfinding around objects as part of the process of obstacle-avoidance. In certain circumstances, however, certain types of obstacles can be "dealt with" in special ways. There are three types of such obstacles (or strategies for by-passing the obstacle): low mid and high threshold. Low-threshold obstacles are ones that are ignored completely at pathfinding time and then dealt with as needed (e.g. smashing and vaulting). This means that even if the object is easily avoided, it will be dealt with regardless. A hunter for example, will not even attempt to avoid the space crate that is in his way. High-threshold obstacles, by contrast, are dealt with only as a last resort. So for example, an elite will destroy a destroyable obstacle in his path only if there is no other convenient way to get to where he wants to go.

Low threshold

- **Trample**: Ignore the obstacle altogether, possibly dealing it damage as you step on it.
- **Smash**: Havok objects that are appropriately annotated (in their object-definition) can be knocked aside by certain types of characters (e.g. hunters smashing aside convenant space crates).
- **Vault**: Objects with appropriate markers can be vaulted over by certain AI characters (e.g. elite vaulting over overturned covenant space crate).
- **Leap**: Obstacle can be leaped over entirely

High threshold

- **Destroyable**: When the destroy_obstacle behavior is enabled, characters can destroy destroyable obstacles if they are uniquely blocking the path to their goal (and there is no convenient way to get around it).
- **Knock**: Some obstacles can be knocked aside, or knocked over, in order to get past them (involves far more effort than merely smashing obstacles).
- **Mount**: Can jump on top of object to get past it.
- **Hoist**: AIs can hoist themselves onto the top of the object to get past it.

Object-Behaviors

- **Destroy cover**: If a target has taken cover behind an object that is destroyable, an AI might respond by attempting to destroy the object.
- **Mount cover**: If a target has taken cover behind a havok object, the AI might decide, if it can, to leap onto the top of the object to expose its target.
- **Hoist to uncover**: A target is uncovered by hoisting onto the cover obstacle.
- **Knock to uncover**

- **Knock to create cover**
- **Knock to deal damage**: a nearby object can be knocked onto the target in order to damage it

AMBIENT LIFE

Flocks

A flock of a certain kind of unit can be placed down for flavor in a certain area. The units making up the flock will be pretty stupid, and so not very fun to interact with. The best way to use them is for flocks of birds in the distance, or flocks of spaceships flying overhead.

A flock is placed through a block in the scenario definition. This block contains all the parameters that control the flock (which can be tweaked to make it look like anything from a loose flock of birds to a tight school of fish). You can also control the number of members in the flock and the noise parameters that can add a small amount of random movement to each flock-member's heading.

Due to run-time considerations, flocks cannot directly perceive the environment or any of the objects in it. Instead the flock's allowable space is defined by a simple geometric shape which is placed by a level designer. Currently the only supported shape is a squashed-sphere shape, which is defined by a center, a radius (xy) and a height (z). The flock will wander about within this shape by turn around when it reaches its edges.

AI Behavior List

11/23/2022 • 8 minutes to read

> **NOTE**
> Behavior in italics are IMPULSES, which means they simply trigger another behavior to run, or run a specialized piece of code – however, they themselves have no duration. As such you will never seem them in the behavior stack of a character.

> **NOTE**
> "Danger" has a very specific technical meaning in Halo 2. You can see the amount of danger an AI thinks he's in by turning on "ai_render_emotions 1" at the console and selecting a character.

> **NOTE**
> Most of the parameters controlling behavior (timing, probabilities, etc.) are located in the .character tag for the character.

GENERAL

- **Root**: Root of behavior tree (always on)
- **Null**: literally, do nothing
- **Obey**: running a command script
- **Guard**: stand on a firing position
- **Ready**: Run upon first transition into a combat state

ENGAGE

Runnable when enemies present

- **Engage**: root of engage subtree
- **Fight**: Stand and shoot. Also handles maneuvering when targets moves outside of ideal combat ranges (based on weapon properties of character tag)
- **Melee charge**: run towards the target and melee
- **Surprise**: When target's back is turned, sneak up behind to close distance (often followed by melee-charge)
- **Grenade impulse**: Consider throwing grenades (probabilisic, and subject to timing constraints. See character tag)
- **Anti-vehicle-grenade**: Throw grenades at vehicles when on foot
- **Stalk**: When given "suppress combat" orders (a flag in the order structure), AI stalks instead of fighting visible target (i.e. get close to them, but don't open fire)

BERSERK

Berserking is a state where the AI behaves extremely aggressively, often throwing down all but melee weapons, and then hard-charging their target.

- **Last man berserk**: when all allies of my type have died in my area, berserk.
- **Stuck-with-plasma-grenade-berserk**: when stuck with a plasma grenade, berserk.

PRESEARCH

When we first lose sight of a target, we consider our "presearch" options.

- **Presearch**: Root of presearch subtree
- **Presearch uncover**: Run the uncover behavior (see SEARCH)
- **Destroy cover**: If target is hiding behind a destroyable object, shoot the object
- **Suppressing fire**: Failing any of the above, stand and shoot at the last place you saw target. If the target moves, this behavior will finish quickly. If not, it will last a longish time, giving the illusion of pinning the target down.
- **Grenade uncover**: throw a grenade to flush the target from behind cover (only after we've just lost sight of them)
- **Leap-on-cover**: if the player is hiding behind an object that has mount or hoist hints on it, jump on top of it.

SEARCH

The target has been gone for some time, or we're just coming out of cover.

- **Search**: root of search subtree
- **Uncover**: pick a firing point from where I expect to be able to see the target that is currently hidden.
- **Investigate**: Go to the point I currently think the target is at.
- **Pursuit-sunc**: If I'm searching with other people, wait for them to investigate their current search point.
- **Pursuit**: Not having found the target where I though it would be, pick another point (current hidden from me) to search at.
- **Postsearch**: We have given up on the target. Pick a point in the open near where we first lost sight of the target.
- **Coverme_investigate**: I'm searching with a friend. Have him cover me (stay put) while I investigate my current search point.

SELF-PRESERVATION

Self-preservation is a set of controlled cover-seeking/defensive behaviors. This is opposed to RETREAT, which is a full-on panic.

- **Self-preservation**: Root of the self-preservation subtree
- **Cover**: Find a point that is hidden from my target (the guy I'm hiding from) and go there, possibly playing a cool lean-up-against-the-wall animation when I get there.
- **Cover-peek**: After I've been in cover a while, if I am leaning against a wall or bunkering, lean out from

behind wall an fire on enemy is they are then visible. Duck back behind cover immediately if I'm hit by enemy fire.

- **Avoid**: If a danger zone is coming my way (grenade, or mortar, or danger enemy vehicle) move to a point that is safe from that danger zone.
- **Evasion impulse**: Hop to the side to avoid enemy fire.
- **Dive impulse**: If I'm in immediate danger from a vehicle or grenade or suck, dive away from it.
- **Danger cover impulse**: If I'm in a certain amount of danger (and my shields are below a certain level, etc.), run self-preserve. (Considered when running ENGAGE).
- **Danger crouch impulse**: When I'm in a certain amount of danger, crouch down.
- **Proximity melee**: If I'm trying to self-preserve, but my target gets too close to me, melee charge him instead of running away.
- **Proximity-self-preserve**: If my target gets too close to me, self preserve (considered when running ENGAGE)
- **Unreachable-enemy-cover**: If I'm being sniped by an enemy from beyond my maximum combat range, take cover.
- **Scary-target-cover**: If my target is facing me and aware of me, and is very scary (e.g. he's a flood juggernaut, or he's holding a scary weapon like a rocket launcher and I'm not), self-preserve (considered when running ENGAGE).
- **Group-emerge**: If I'm finished covering and there are other guys near me who are also covering, wait for them so that we can all emerge from cover together.

RETREAT

A paniced, mindless retreat from a target

- **Retreat**: Root of retreat subtree
- **Retreat-grenade**: Throw a grenade at my target before taking off.
- **Flee**: Run away to a hiding point, playing my crazy fleeing animations as I go.
- **Cower**: Crouch down at my hiding place, fearing for my life.
- **Low-shield-retreat**: Considered when running ENGAGE
- **Scary target retreat**: Considered when running ENGAGE
- **Leader dead retreat**: GRUNTS ONLY. An elite died, and there are no more elites around me. Flee!
- **Peer dead retreat**: Probabilistically retreat is one my friends just died.
- **Danger retreat**: Above a certain danger level, possibly retreat.
- **Proximity retreat**: If my target gets too close to me, retreat.
- **Charge when cornered**: If I have nowhere else to go, turn around and run melee-charge.
- **Surprise retreat**: If I'm surprised by a target, possibly retreat.
- **Overheated weapon retreat**: If my weapon overheated, retreat!

AMBUSH

After a retreat target has cowered for enough time, he begins to try to ambush his target (i.e. emerge at an opportune time)

- **Ambush**: Root of ambush subtree
- **Coordinated ambush**: If there are others trying to ambush, hook up with them, and ambush together.
- **Proximity ambush**: If my target appears very near to me, start fighting again.
- **Vulernable enemy ambush**: My target is vulnerable, e.g. nearby and facing away from me. Ambush.
- **Nowhere to run ambush**: I have nowhere else to run, start fighting again.

VEHICLE

This section covers behaviors for getting in/out of vehicles, and things to do when you're in a vehicle.

- **Vehicle**: Get in a vehicle.
- **Enter friendly vehicle**: If the player is in a nearby vehicle, get in with him.
- **Re-enter flipped vehicle**: A vehicle that I was recently flipped out of is upright again, so get back into it.
- **Vehicle entry engage impulse**: I'm fighting an enemy, and there's a vehicle available nearby. Get in it.
- **Vehicle board**: Jump on my enemy's vehicle and board him.
- **Vehicle fight**: Drift around my target and fight.
- **Vehicle charge**: Strafe my target
- **Vehicle ram**: Ram my target.
- **Vehicle cover**: Pick a point far away from my target (but preferably still visible) and go there.
- **Damage vehicle cover**: Run vehicle cover if my vehicle has taken a certain amount of damage recently.
- **Exposed rear cover**: Someone is shooting me from behind. Run vehicle-cover.
- **Vehicle avoid**: Danger is coming my way. Strafe to the side to avoid it.
- **Vehicle pickup**: I need a gunner. Go find someone to be my gunner.
- **No driver exit vehicle impulse**: I'm in a vehicle without a driver. If I'm a passenger get out immediately.
- **Danger vehicle exit impulse**: I'm in a vehicle without a driver, and danger is heading my way. Get out.
- **Vehicle flip**: A vehicle that I want to get into is flipped nearby. Either flip it myself (if it's ghost-sized) or get two buddies to help me flip it (if it's warthog-sized).
- **Vehicle turtle**: (Wraiths) "Crouch" down if I've taken a certain amount of damage.

POSTCOMBAT

What we do after combat is finished and there are no more enemies.

- **Postcombat**: root of postcombat subtree
- **Post-postcombat**: root of postpostcombat subtree
- **Check friend**: Kneel down beside the body of a friend and say something touching.
- **Shoot corpse**: Shoot the dead body of an enemy corpse.

- **Postcombat approach**: In postpostcombat, regroup with my friends, if we're scattered.

ALERT

When we THINK there are enemies in the area, but we don't know where they are (either because we were fighting them and they got away or because the designers has told us we should be alert).

- **Alert**: Root of alert subtree

IDLE

There are no enemies around.

- **Idle**: root of idle subtree
- **Wander**: We have a small area to available to us. Wander from one firing point to another.
- **Flight wander**: Wander for flying creatures
- **Patrol**: Walk around, picking a firing position in each area in my orders.
- **Fall-asleep**: If nothing is going on, go to sleep.

SWARMS/FLOOD

Swarm- and flood-specific behaviors. Swarms have their own tree entirely separate from any other type of character.

- **Swarm root**
- **Swarm attack**: Run toward a target and jump at it.
- **Support attack**: If a flood combat form is fighting the target, hang back.
- **Infect**: Run up to a dead flood body on the ground and resurrect it.
- **Scatter**: If a certain percentage of a swarm has died recently, run away.
- **Eject parasite**: If a flood combat form loses both its arms, it releases an infection form and collapses.
- **Flood self-preservation**: All flood retreat at the same time.
- **Juggernaut flurry**: After receiving a certain amount of damage, the juggernaut runs a "flurry" melee animation (undirected area melee damage)

SENTINELS

- **Enforcer weapon control**: Enforcer changes the weapon it uses depending on target range, suppressing fire etc.
- **Grapple**: If target is in a vehicle, boost over the vehicle.

SPECIAL

- **Formation**: Jackals can form a number of formations and march forward together.
- **Grunt scare by elite**: A grunt walks up to an elite, the elite turns around and plays his "berserk" animation. The grunt then flees from him.
- **Stunned**: Get stunned by a nearby grenade explosion. Stumble around, clutching your ears.

- **Cure isolation**: I'm standing on a nonwalkable sector, or an isolated walkable sector. Try a bunch of tricks to get back to a walkable surface.
- **Deploy turret**: If I'm carrying a deployable turret, run to a specially-marked script point and deploy it there

AI Engineering Outline

11/23/2022 • 6 minutes to read

Spatial Encoding

Zones

Replacing Encounters will be a "Zone" system. Like encounters, Zones will contain groups of firing points, which actors will use to control their movement. Unlike encounters, Zones are not team-specific, so the spatial information they encode can be used by any actor.

Areas

Zones will contain Areas, groups of firing points that cover individual regions within a single zone. These areas will be used for the purposes of scripting and order-specification (see below). Potentially, these firing points could be jittered, to allow some variation in position selection (and to avoid the problem where two actors conflict over the same position).

While defined as a set of points, areas also have an associated "margin of error", which denotes a radius around each point within which all space is still considered to be part of the area. In other words, an actor within a margin distance X of one of the firing points of an area is still considered to be within that area. Areas can be used to define an "allowable region" within which an actor can operate (this would be done with the actors orders, e.g. "defend this region and under no circumstances can you leave it").

Firing Point Generation

In addition to the statically laid firing points provided by Areas, actors will be able to generate new firing points for consideration on the fly. These will almost always be task-specific. Thus, if an actor is looking for cover, it will generate firing points around nearby obstacles. These generated points might also be filtered by the agent's allowable region. Thus an actor might be allowed to take cover behind an obstacle as long as it is within his allowable region.

Firing points will not be generated around static obstacles, since designers can place them when creating the levels. However, generating them around dynamic obstacles will allow the actor to take cover behind vehicles, "blocks" (dynamic obstacles that can be meleed to create cover) and each other. There might even be some rudimentary planning possible in this respect – an object offering some cover points but "only if you melee me first". These dynamic firing points could also be used in establishing attack formations and setting up positions for coordinated actions.

Zone-level Activation Control

Actor activation will be controlled on a zone-by-zone basis. Based on what clusters are potentially visible to the viewer, all the zones that are not visible (do not impinge on the PVS) will be deactivated. Consequently all squads in the zone will be deactivated until the player returns to the zone.

Some allowance might be made for squad positions – a squad on their way to an active zone, but not there yet, for example, might be allowed to remain active.

Squads

Since Encounters are being deprecated, squads will move out of the Encounter structure and become a Scenario-level representation. Actors will generally go into leaf-node Squads, while these leaves will be held together in a larger structure by Squad-Groups.

The main difference with the new representation will be that squads will be put into a hierarchical structure,

rather than just a list. This will allow orders, for example, to be given to a potentially large number of actors, while still being able to give smaller groups more specific instructions.

Squad_groups

The military hierarchy will be encoded in a hierarchical structure, however, only leaf-nodes in this tree will be allowed to actually contain actors. To avoid confusion, intermediate (branch) nodes will be called *squad_groups* (represented in code by a squad_group struct). Squad_groups, like squads, will maintain statistics (current membership, initial membership, etc.) encompassing all their children.

Orders

Some of the burden of scripting complex battle sequences will be taken over by the new Order system. Orders are atomic instructions to the actors of a squad – instructions on the order of "defend this position", "secure this area", and so on. Orders are simple associations of a location and a style. Possible styles include "idle", "defend" and "attack" (more styles to come). When an order is given to a group of actors (through their *Squad* grouping, see below) the actors will do their best to comply with it.

Orders have an ending condition, indicating when the order has been successfully completed. Possible ending conditions include:

- **If [greater than / less than] X actors alive in squad**
- **If squad at X strength**
- **If enemy sighted**
- **If unopposable enemy within X distance**
- **After X ticks**
- **If alerted by squad X**
- **Arbitrary condition**: specified by script.

Any combination of these conditions can be used to trigger the end of an order. A set of ending conditions also contains a reference to a next order, to which the squad will switch when an order is completed. An order can contain multiple sets of ending conditions (and hence can be followed by multiple orders) reflecting the different possible outcomes of a battle.

These orders, linked together as they are through their ending conditions form an *order tree*. Scripting commands will be available to impose a change of orders (and perhaps a change of order-trees).

There is special interaction between the order system and the squad-tree. When an order is given to a top-level node in the squad-tree, the order filters down to all its children. The exception: an order may be "locked", such that no parent-level orders can override it. When the ending-conditions of a locked order are met, the next-order is activated (with its lock-status used). If no next-action is provided, the squads will default to the nearest ancestor's order.

Order options include

- **Locking**: an order cannot be overriden (by an order to a parent of the squad in question, see below). The only ways for a locked order to be changed is by satisfying a set of ending conditions or by an explicit script call.
- **Always active**: A squad executing the command should be be always be kept active.
- **Debug**: Causes additional debugging information to be output when the order is started/stopped.

Hints

Information is also available through special designer-provided hints. These hints provide information that would be too expensive (or impossible) to derive at run-time, such as good locations to snipe from, or what direction to retreat in (dependant on the subtle ebb and flow of the battle). They might also include information such as "this is a good place to jump up and shoot at the enemy" or "this is something that you could jump over".

Hints can be associated both with either an order or an entire zone. The former is for team-specific hints (e.g. "flee in that direction") while the zone-hints are generally available knowledge (e.g. "that is a good position to snipe from").

Possible hint-types include:

- **Firing position**
- **Cover position**
- **Concealment position**
- **Search position**
- **Sniping position**
- **Climbable**
- **Hang down and shoot from me**
- **Interesting-look**
- **Interesting-touch**
- **Interesting-destroy**
- **No-stopping zone**
- **No-travel zone**
- **Vaulting/jumping**

Like orders, hints express themselves in terms of *Areas* (see above), i.e. in terms of one or a set of firing points (possible with some designer-provided margin in the area-definition).

Hints can also be attached to specific objects (especially if that object is moveable) and object-types (so that the designer need not place a copy of the same hint on every tree, for example). Thus the pinball machine, for example, will tell the actor "I can help you with your entertainment drive".

Player-Modeling

It would interesting to try to map a brief history of the player's recent behavior to a discrete style. Thus, the player might be perceived to be in "sniper-mode", "aggressive mode", and so on. The game could react to this mode through specific vocalizations, specialized battle set-ups, and appropriate team (and enemy) behavior. The player mode would be available through script as well as being a possible ending-condition for orders.

AI Pathfinding

11/23/2022 • 5 minutes to read

ORGANIZATION

Sectors

In contrast to Halo 1, where path-finding took place directly on the collision BSP surfaces, for Halo 2, we will be grouping surfaces into sectors. These sectors will be convex pathfindable areas within which completely reactive navigation will suffice to get the actors from any point in the sector to any other point in the sector. Sectors will likely be larger than bsp surfaces, and will be able to contain uneven terrain (i.e. normals need not be near-equal) – as long as the terrain is not SO uneven that it can't be traversed in all directions. Note also that while sectors will be created, for example, for climbable (non-walkable) surfaces, no sectors can COMBINE walkable and non-walkable surfaces (for the same reason as above, that an actor that cannot climb will not be able to traverse the sector-space freely). Similarly, breakable and non-breakable surfaces cannot be combined.

Initially, in most cases, sectors will be groupings of existing BSP surfaces – i.e. each collision surface will belong to exactly one sector. This restriction might be eased later on if it is deemed necessary in some cases to split individual surfaces in order to make the overall sector partitioning cleaner. ALSO, note that the environmental collision BSP will NOT be the only source of sectors – certain environment objects will also be able to be walked on – sectors on these objects will be created with similar rules.

Links

Sectors will be connected via links (the equivalent of edges in the BSP surface rep). Again, most links will simply be references to some underlying collision BSP edge (since in the base case the sector-reps will be groupings of collision surfaces).

Most of these links will simply represent the need for an actor to walk from one sector to another (around a wall, for example). However, in some cases, special-purpose information might be encoded in the link to indicate that the two sectors are connected in a special way – for example, the actor needs to climb a ladder to get from one sector to another, or the actor needs to jump.

See section below on "special-purpose sector-links" for more details.

ALGORITHMS

Initially the algorithms for path-planning will remain unchanged, with the exception of the new terrain-preprocessing step.

A*

The fundamental path-planning algorithm remains A*.

Terrain pre-processing

The fundamental new algorithm, the terrain pre-processing step requires us to generate a list of sectors based in large part on the collision bsp model. The difference, again, is sectors don't require surfaces of nearly-equal normals – they can incorporate fairly uneven surfaces, as long as the surface is still pathfindable in all directions.

We will use the standard greedy algorithm done in Halo1 for the collision BSP and in the Tozour article. We will repeatedly iterate over the collision surfaces, attempting to coallesce them into larger convex sectors (on subsequent iterations, sectors will be coallesced and so on, until no more sectors can be merged). Depending on

what the results of this process produces, we may well also perform something like the 3→ 2 merging process described in the Tozour article (unfortunately, this would create edges and vertices in the path-finding graph that do NOT exist in the collision BSP – resulting in larger storage requirements for those edges and vertices).

Surface → Sector Mapping

The process of discovering the sector at a certain point is critical to allowing AIs to pathfind about their environment. Since AIs always have a current surface, we can use a surface à sector mapping. We do this by laying down 2-d bsps on the collision surfaces at sector-creation time. These bsps can then be queried with a surface point to find the sector.

SPECIAL-PURPOSE SECTOR-LINKS

Link types

Beyond simply walking from one sector to another, sometimes a more specific behavior is required. What type of behavior depends on information that will be stored in the link (quite a bit of it is placed by hand by a designer or environment artist in the form of environmental "hints"). Envisioned link-types include:

- Elevators
- Ladders
- Jump points
- Climb points
- Vault points
- Crawl points
- ...

Behavior-system integration

In order to incorporate a behavioral component into individual path-links, the command-list notion of an "AI atom" will be recycled. The "atom-evaluator", the piece of codes which transforms atoms into specific orders for the actor to execute, will be pulled out of the action_obey code, and be made a general module that can be used by any ai system (e.g. individual actions could conceivably express their effects in terms of atoms). Links in the pathfinding graph can contain one or a number of these AI atoms, which together specify the type of behavior required to traverse a link. For example, an "elevator" link, might contain the following atoms:

- Navigate to point <x,y,z>
- Orient to direction <x,y,z>
- play animation "push button"
- Wait until elevator arrives (until event detection)
- navigate to <x,y,z>
- Wait until elevator arrives at destination
- navigate to point <x,y,z>

Even a "normal" link – the kind whereby one can simply walk from one sector to another – could be considered to contain a single atom of "Navigate to <x,y,z>" where the point specified is a point on the edge between the two sectors (though it's unlikely that it will actually be implemented this way).

DYNAMIC SURFACES

A number of the path-findable surfaces will be attached to dynamic environment objects (and vehicles and so on) which means that many of the links they have to static sectors will be temporary. For example, a warthog being driven through a scene might have a small sector (probably a single point) where enemies may jump onto its hood to attack the driver. As the jeep drives from sector to sector, an enemy must know which sector it has to navigate to to jump onto the jeep.

This might be accompished as follows: a “jump hint” on the hood of the warthog would need to be placed by the artist or designer of the vehicle. This jump hint would contain information both on where on the hood the enemy would land but also a small number of locations on the ground (in warthog-relative space) that the enemy could jump from. As the warthog drives around, these points follow it. These points can also be traced to a specific sector. Thus, if an enemy wish to jump onto the warthog, it has to navigate first to the sector from which it can begin its jump.

The link encoding the ability to jump onto the warthog hood must obviously be invalidated if the warthog is overturned.

Halo 2 Art Home

11/23/2022 • 2 minutes to read

More information on Art related topics for Halo 2 can be found below:

Animation Home

Additional information for Animation in Halo 2 modding.

Bitmaps Home

A bitmap is any image file used in the game.

Decorators Home

The decorator system is used to create a large number of (relatively) low-cost objects in an environment, such as grass or rock.

Decals Home

Additional information for Decals in Halo 2 modding.

Crates Home

Crate tags are used for passive objects that require Havok simulation (that is, objects that can be moved in the game - like crates).

Animations

11/23/2022 • 4 minutes to read

USEFUL TOOL COMMANDS

Tool commands useful for animators.

Compile Model animations

A set of animation source files can be compiled into a model_animation_graph tag.

```
# model-animations <source-directory> [flags]
tool model-animations "objects\characters\masterchief"
```

- source-directory - A local data path to the root of a model source directory.
- flags

For the example above, Tool would expect to find a set of corresponding animation source files at data\objects\characters\masterchief\animations. *Assuming no errors, it would be compiled into* tags\objects\characters\masterchief\masterchief.model_animation_graph.

Get Model animation count

Goes to all the animations in a directory and child directories. Once it's done it prints debug info.

```
# model-animation-count <source-directory>
tool model-animation-count "objects\characters\masterchief"
```

- source-directory - A local tag path to a directory containing model_animation_graph tags or child folders with model_animation_graph tags.

Monitor Model animation status

Goes to all the animations in a directory and child directories. Once it's done it prints debug info.

```
# model-animation-status <source-directory>
tool model-animation-status "objects\characters\masterchief"
```

- source-directory - A local tag path to a directory containing model_animation_graph tags or child folders with model_animation_graph tags.

USEFUL DEBUG COMMANDS

Useful debug commands for animators.

DEBUG OBJECT UNIT STATE

Add these lines to the **init.txt** file in your Halo 2 working folder.

- debug_objects on
- debug_objects_unit_vectors off
- debug_objects_bounding_spheres off

- debug_objects_collision_models off
- debug_objects_physics_models off
- debug_objects_unit_state on

These commands turn on debug_object rendering, then turn off a bunch of output you don't need, and finally turn on the one mode you do need (debug_object_unit_state).

REFERENCE IMAGE	NOTES
	Above the head of each unit, the following data is written out: mode= [seat][weapon class][weapon type][state] ← the state of the unit as far as the AI is concerned base = [name of the base animation being played] aim = [name of the aiming screen is use] look = [name of the looking screen in use] over = [name of the current overlay] ← firing, reloads, etc...

DEBUG FIRST PERSON ANIMATIONS

You can view which first person animations are playing by typing in the console command:

debug_first_person_weapons on

If the HUD is getting in the way, you can turn it off by typing in the console command:

show_hud off

REFERENCE IMAGE	NOTES
	In the upper left quadrant of the screen, the following data is written out: mode= [default]:[weapon class]:[weapon type]:[mode] base = base animation moving (if moving) = moving animation overlay = overlay animation pitch/turn = pitch/turn overlay animation

MODEL AND ANIMATION NAME PROCESSING

When models and animations get imported into the game, the names are first processed:

1. All uppercase letters are changed to lowercase
2. All white spaces and dashes [-] are replaced with underscores [_]
3. Prefixes for skeletons are removed along with proceeding underscores

- frame
- bone
- bip01
- b

4. Strings are terminated at "*atr*"
 - ...*atr*... → ...\0atr...
5. Attributes are processed
 - characters following "*atr*" each identify an attribute (not including characters considered white space or [_]), ie u = unused

> **NOTE**
> All attributes should be set in Max or Maya. Properly named nodes - ie, "_atr_u" - are not imported into Game.

> **IMPORTANT**
> NAMED NODES ARE PREFERRED TO USER-DEFINED PROPERTIES IN MAX

MONITORING CHANGES IN ANIMATIONS WITH ASSOCIATED SOUNDS

If you change an animation which has sounds attached to it, you will be warned by Tool and a flag will be set in the animation showing it has been changed.

The two flags found in each animation are:

"do not monitor changes" -> when set, changes in the animation are ignored, even if audio is attached "verify sound events" -> set when the animation has been changed, and sound events are attached

The “verify sound events” flag will be set for any animation that has sounds attached whenever the duration, node list, or animation content is changed. If you are simply re-importing the graph without changing any animations, the flags will not be set.

For example, assume after re-importing the elite graph, you got the following result:

```
[WARNING] Animation 'stand:pistol:land_hard' has changed . Its sound events must be checked.
[WARNING] Animation 'stand:pistol:land_soft' has changed. Its sound events must be checked.
[WARNING] Animation 'stand:pistol:melee' has changed. Its sound events must be checked.
[WARNING] Animation 'stand:pistol:move_back' has changed. Its sound events must be checked.
[WARNING] Animation 'stand:pistol:move_front' has changed. Its sound events must be checked.
[WARNING] Animation 'stand:pistol:move_left' has changed. Its sound events must be checked.
..etc..
```

This tells you that a bunch of animations with sound events have changed, and the sound events need to be verified. Once the sound events have been checked, the “verify sound events” flag can be cleared. Whenever possible, animators should check to make sure they have not changed the animation in a way that will screw the sound events attached. If everything is ok, clear the flag. Ideally, the only reason to check in a file with the “verify sound events” flag set is if new audio has to be authored to match your changed animation.

You can monitor these flags by using the new tool command model-animation-status as in:

tool model-animation-status objects\characters

This will produce a list of all animations in the directory requested (and all sub-directories) which have the "verify sound events" flag set. You can also pump the output into a text file using the > symbol. As in:

tool model-animation-status objects\characters >output.txt

If you want to list the results of ALL animation graphs, use a set of empty quotes for the directory

tool model-animation-status ">output.txt

Here's an example of the list generated:

```
objects\characters\elite\elite => stand:pistol:land_hard
objects\characters\elite\elite => stand:pistol:land_soft
objects\characters\elite\elite => stand:pistol:melee
objects\characters\elite\elite => stand:pistol:move_back
objects\characters\elite\elite => stand:pistol:move_front
objects\characters\elite\elite => stand:pistol:move_left
Finished. 6 changed animations found
```

Each entry is graph_name => animation_name

More information on animations can be found in these additional pages.

Damage Animations

First Person Animations

Inherited Animations

Animations - Damage Animations

11/23/2022 • 2 minutes to read

Damage animations are identified by **type**, **direction**, and **region**. So, if you made every permutation of damage, region and type you'd have 176 damage animations per character.

Damage Types

There are four kinds of damage animations.

- **soft_ping** - overlay (jmo) which shows impact from light damage
- **hard_ping** - state animation (jma) which shows the impact of heavy damage (usually a stagger or side-step)
- **soft_kill** - state_animation (jma) which shows death from light damage
- **hard_kill** - state_animation (jma) which shows death from heavy damage

Damage Directions

- **_animation_damage_direction_front**
- **_animation_damage_direction_left**
- **_animation_damage_direction_right**
- **_animation_damage_direction_back**

Damage Regions

- **_damage_part_gut,**
- **_damage_part_chest,**
- **_damage_part_head,**
- **_damage_part_l-arm,**
- **_damage_part_l-hand,**
- **_damage_part_l-leg,**
- **_damage_part_l-foot,**
- **_damage_part_r-arm,**
- **_damage_part_r-hand,**
- **_damage_part_r-leg,**
- **_damage_part_r-foot,**

Damage Animation Inheritence

Since we can't obviously author that many individual animations, there is an inheritance scheme in place which lets the engine find the closest possible animation to play. It uses a built-in search order to find an animation it

can play.

The searching works like this:

Suppose an [h_ping, back, right_shoulder] animation is desired, but one does not exist. The engine will first determine if any h-pings from the back exist. If there are no other h_pings available from the back, the other directions are searched until a set of h_pings is found. The search order is:

CURRENT DIRECTION	NEXT DIRECTION TO TRY
_animation_damage_direction_front	_animation_damage_direction_left _animation_damage_direction_back _animation_damage_direction_right
_animation_damage_direction_right	_animation_damage_direction_front

Once a set of hard pings is found, the damage regions are searched to find the closest match. The region order is:

CURRENT REGION	NEXT REGION TO TRY
_damage_part_gut	_damage_part_head
_damage_part_chest	_damage_part_gut
_damage_part_head	_damage_part_chest
_damage_part_l-arm	_damage_part_chest
_damage_part_l-hand	_damage_part_l-arm
_damage_part_l-leg	_damage_part_gut
_damage_part_l-foot	_damage_part_l-leg
_damage_part_r-arm	_damage_part_chest
_damage_part_r-hand	_damage_part_r-arm
_damage_part_r-leg	_damage_part_gut
_damage_part_r-foot	_damage_part_r-leg

Importing Damage Animations

Currently, to import a damage animation through Tool, it must be named in the format

[type][direction][region] as in:

“h_ping front gut.jma”

For this system to work, at least one gut, chest or head animation must be supplied for each damage type.

For example, you could supply these four animations and have a working system:

```
“s_ping front gut.jma”
“h_ping back chest.jma”
“s_kill front gut.jma”
“h_kill back head.jma”
```

...because these four animations provide at least one head, chest or gut animation for each of the four damage types. The directions used don’t matter in this minimal case.

Animations - First Person Animations

11/23/2022 • 3 minutes to read

Below is an outline of the files that comprise the Halo 2 first person animation system.

BASIC ANIMATIONS

The following files is the base file list you need for a standard first person weapon animation set. They should reside in the weapon's "fp" animation folder.

For instance you would find the SMG first person animations here:
\data\objects\weapons\rifle\smg\fp_smg\animations

We recommend creating a "working" directory in the "animations" folder where you can store your DCC files and reference.

- **first-person firing.jmo**
- **first-person idle.jmm**
- **first-person melee-1sthit.jmm**
- **first-person melee-1sthit2i.jmm**
- **first-person melee-2ndhit.jmm**
- **first-person melee-2ndhit2i.jmm**
- **first-person melee-3rdhit.jmm**
- **first-person moving.jmo**
- **first-person overlays.jmo**
- **first-person posing.jmm**
- **first-person put-away.jmm**
- **first-person ready.jmm**
- **first-person reload-full.jmm**
- **first-person reload-empty.jmm**
- **first-person throw-grenade.jmm**
- **rename.txt**

FIRST PERSON ANIMATION BREAKDOWN

The following text describe the purpose for each animation file needed for a basic first person animation graph.

first-person firing.jmo

- firing animation overlay
- shares timing with the 3rd person firing animation for that weapon

first-person idle.jmm

- basic breath idle
- should have at least one other variation

first-person melee-1sthit.jmm

- melee attack
- 1st hit of melee combo
- roughly timed the same as hit #2
- animated from idle pose to end position of attack #1
- hold in end position

first-person melee-1sthit2i.jmm

- return to idle from melee attack #1
- quick return to idle
- used if combo is interrupted or discontinued

first-person melee-2ndhit.jmm

- melee attack
- 2nd hit of melee combo
- roughly timed the same as hit #1
- animated from end position of attack #1 to end position of attack #2
- hold in end position

first-person melee-2ndhit2i.jmm

- return to idle from melee attack #2
- quick return to idle
- used if combo is interrupted or discontinued

first-person melee-3rdhit.jmm

- melee attack
- 3rd hit of melee combo
- biggest attack of the combo
- animated from end position of attack #2 to idle
- returns to idle

first-person moving.jmo

- simulates the character's upper body movement
- gun and arms sway from side to side to simulate footsteps being taken

first-person overlays.jmo

- overlays used to simulate delay/overlapping action

- eg: a character strafes left and the gun/arms delay then move left with the character
- 0-9 [10 frames]

FRAME	NAME	POSE REFERENCE
0	All-Base	- base pose holding gun
1	Arms-Back	- arms move back
2	Arms-Front	- arms move forward
3	Arms-Left	- arms move left
4	Arms-Right	- arms move right
5	All-Base	- base pose holding gun
6	Gun-Left	- gun pitches left
7	Gun-Right	- gun pitches right
8	Gun-Up	- gun pitches up
9	Gun-Down	- gun pitches down (helpful to slide the rig back a little on this one)

first-person posing.jmm

- flavour animation / flavour idle
- used when character is idle for a longer period of time
- eg: character fidgeting with gun
- animated from idle pose to idle pose

first-person put-away.jmm

- used to stow weapon
- weapon completely leaves screen
- animated from idle pose to off screen

first-person ready.jmm

- used to bring weapon to idle pose
- weapon comes from off of screen
- animated from off of screen to idle pose

first-person reload-full.jmm

- reload weapon
- changing magazines/clips/etc
- different for each weapon [eg: shotgun]

first-person reload-empty.jmm

- reload weapon
- reused from "first-person reload-full" animation in the rename.txt

first-person throw-grenade.jmm

- throw grenade
- left handed throw
- do not export with a grenade in the scene

rename.txt

- a text file
- intended to list reused animations

HELPFUL HINTS

> **NOTE**
> export is basically the same method, with the graph living under the weapons' tag directory

> **NOTE**
> start with a static animation as an idle to get the pose down on screen before moving on to creating the rest of the moveset

> **NOTE**
> you can reuse a lot by referring to existing animations as a base to start from

> **NOTE**
> the slightest movement can make or break a first person animation, so be aware of what you are doing and refer to the animations in the engine for the final look

Animations - Inherited Animations

11/23/2022 • 2 minutes to read

Changes to the animation tags

Model_animation_graph tags must always contain the same list of nodes as the model tag. This is a requirement so that we can build lookup tables to map inherited graphs to the local nodes without having knowledge of the model. We perform the actual inheritance while post-processing the animation tag, at which point we no longer have knowledge of the model.

On import, we populate the animation graph's node list with the nodes from the model – not the animation. We no longer compare checksums between model and animation nodes either, since we will allow animations to contain only a subset of model nodes. Instead, we check that all animation nodes appear in the model, and have identical parent nodes. We do all these comparisons as string compares of the node names. If a node can not be found, or it does not have the same parent (with an exception for the root node in the animation), we consider the model and animation incompatible and abort.

Animation data is expanded to contain 4 bit fields (up from 3). The three existing bit fields flag which nodes have animated rotation, translation and scale data. When these bits are not set, data is pulled from the animations 'default' data section. To further reduce the tag size, and facilitate animations which contain only a subset of the model nodes, we will add a 4th bit field which signifies whether a non-animated node has a local default orientation, or whether the default orientation should be read from the model itself.

At run-time, we pre-seed the model orientation list with the default orientations from the model. Then, we only need to overwrite the portions of the data for nodes which are present in the animation.

Inheritance

Each animation graph contains a single link to an inherited tag. At post-process time, we build a run-time list of all inherited graphs by recursively loading each linked tag in the chain. There is a max inheritance list of 8 for the time being. For each inherited tag, we build a lookup table which maps our local node indices to the indices of the inherited graph. Local nodes which do not appear in the inherited graph have an entry of NONE. When computing orientations for a model, we use this table to map model nodes to animation nodes. When the entry is NONE, we can skip the node since we already pre-seeded the orientation list with default model orientations for each node.

Bitmaps

11/23/2022 • 5 minutes to read

A bitmap is any image file used in the game. On the data side, the image file must be in a TIF (.tif) format. It gets converted into a .bitmap tag which can be used in a variety of ways in the game - 2D textures, 3D textures (height or bump maps), cube maps (environment maps), sprites or interface bitmaps.

Bitmap data must be:

- TIF format
- dimensions in powers of two except sprites and interface bitmaps
- does not have to be square (ie, can be 32x512 or any other dimensions that are powers of two)
- 4x4 pixels or greater (2x2 causes Tool to assert)
- no layers
- in folder named “bitmaps”
- other types of bitmaps (.psd, .jpg, etc) will be ignored
- subfolders will be ignored as well as the contents (even if it is a .tif file)

Bitmap tags must be created before the type, compression or usage can be set. You can create a new tag in Guerilla or you can simply import the bitmap and then you can set it's attributes. Either way, you then have to import (or re-import) the bitmaps for any of the settings to take effect.

- **The settings in the tag only affect the bitmap as it is imported; it has no effect on a tag except at time of import.**

Unlike other Tool commands, you must specify the actual bitmaps directory and not just the parent folder. This is because there can be multiple bitmaps in a folder and you want them to retain their own unique names.

BITMAP TYPES OVERVIEW

2D Textures - Ordinary image file for textures and height maps (sometimes called bump or normal maps).

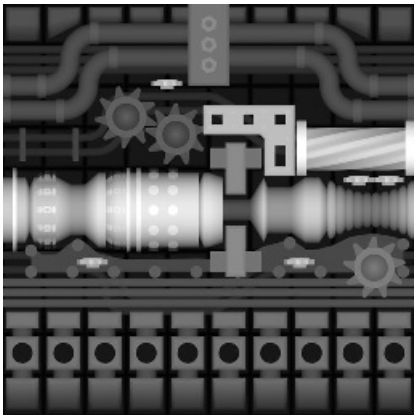


Fig 1. Flat 2D Texture

3D Textures - Used for plasma effects and spherical light falloff. 3d textures are a “stack” of bitmaps that describe a volume:



Fig 2. "stacked" 3D Textures

Cube Maps - Environment or reflection maps.

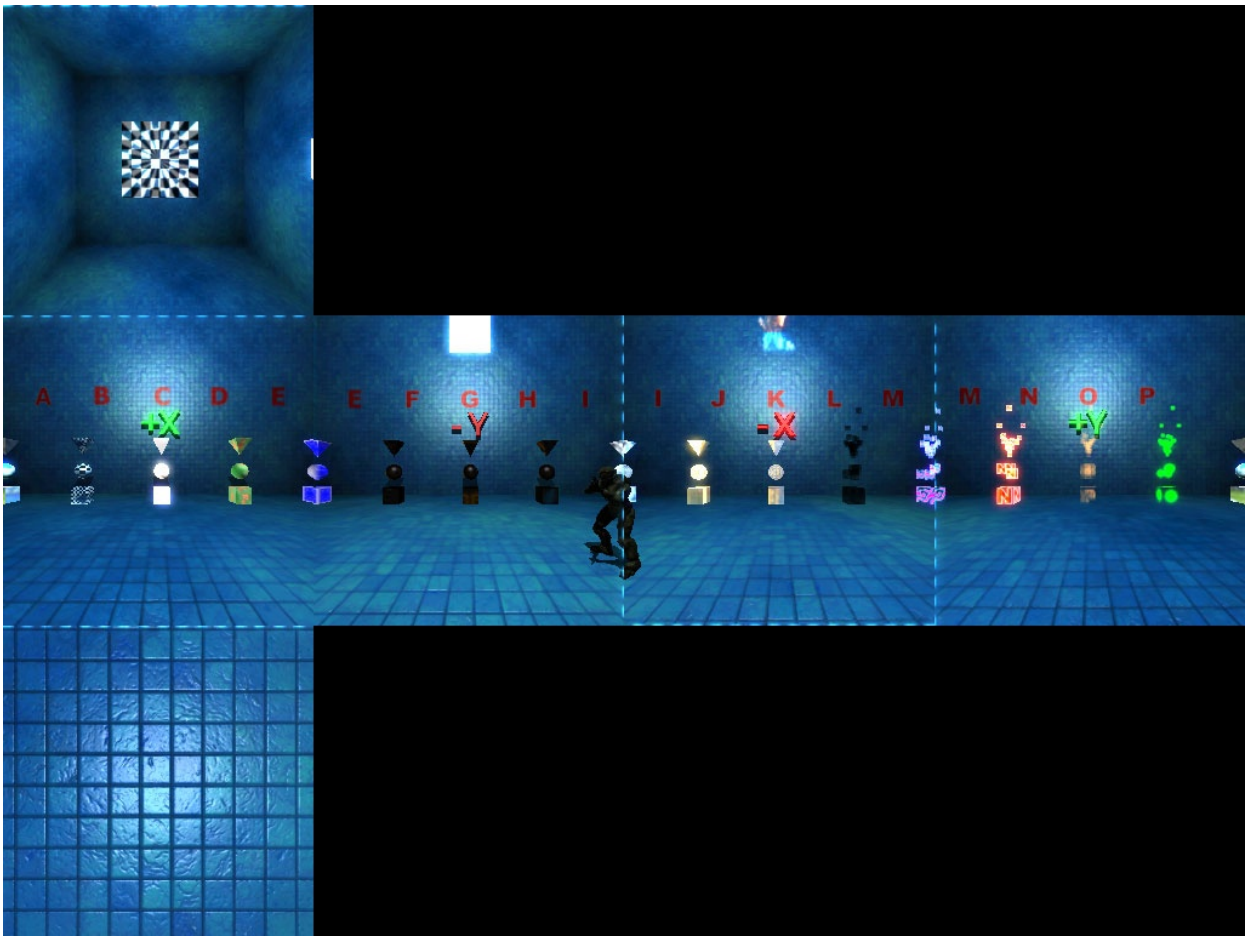


Fig 3. A cubemap.

Sprites - Sequence of images for creating bitmap 'movies' - usually used in effects.

Interface Bitmaps - Bitmaps used in the UI.

COMPRESSION

Bitmaps are compressed using one of six compression formats. Each one has benefits and drawbacks. You want to have the smallest file size possible while still achieving the intended visual effect.

DXT Compression

In all of the DXT compression schemes the colors are compressed the same way - the only difference is in how the alpha channel is dealt with. DXT compressed textures are divided into 4x4 pixel blocks and then each block is reduced independantly to two colors. Those two colors are combined to create (as closely as possible) each of the 16 pixels in the block.

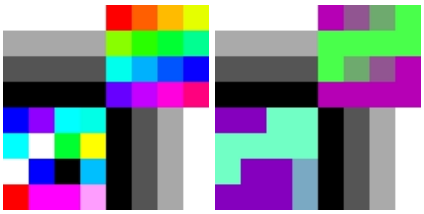


Fig 4. Original 8 x 8 pixel .tif (left) and DXT compressed (right).

Compression Examples

Here's the sample file used in the following examples:

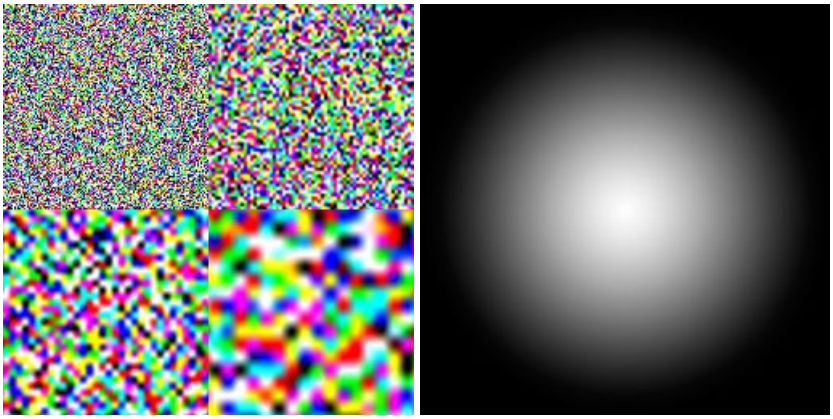


Fig 5. original .tif file - RGB (left) and Alpha (right) **250kb**

Compressed with color-key transparency: DXT1

With DXT1 the alpha channel is compressed to simple black and white and used to create a key color (0) in the RGB. This means that anywhere the alpha = 0 (black), the diffuse color = 0, too.

DXT1 is good for solid blocks of simple colors without alpha information.

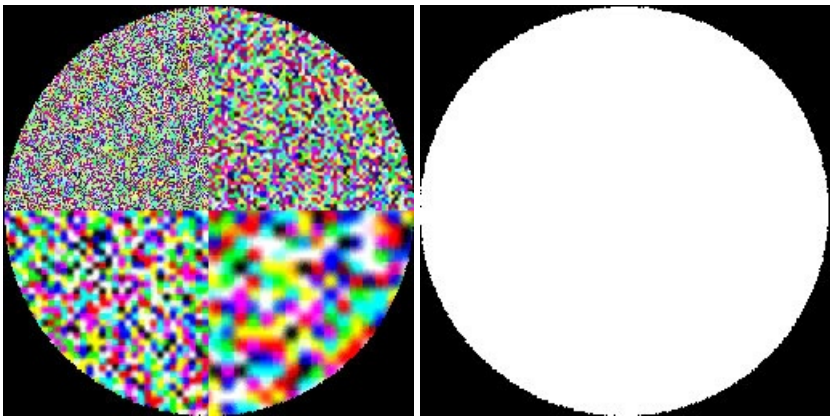


Fig 6. with DXT1 compression 42kb

> **NOTE**
> Anywhere the alpha channel = 0, the diffuse channel will be BLACK.

Compressed with explicit alpha: DXT2/3

DXT2/3 adds 4-bit (16 color) alpha to DXT color compression (see above).

DXT2/3 is good for solid blocks of simple colors with sharply contrasting alpha information.

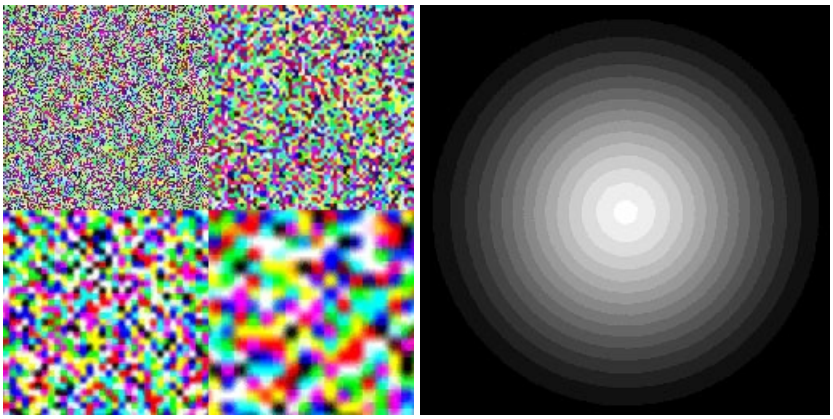


Fig 7. with DXT2/3 compression 85kb

Compressed with interpolated alpha: DXT4/5

DXT4/5 adds 4-bit gradated alpha to DXT color compression (see above). It uses 3-bit alpha (8 colors) and then gradates between those colors using a fourth bit.

DXT4/5 is good for solid blocks of simple colors with smoothly gradated alpha information.

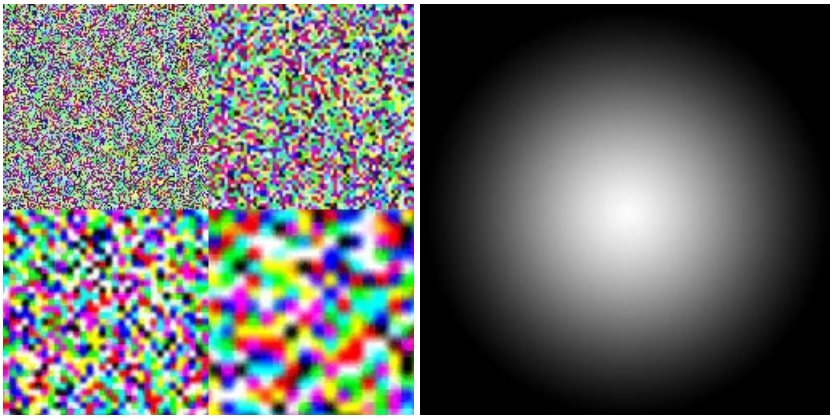


Fig 8. with DXT4/5 compression 85kb

16-bit color

Uses 16 bits per pixel. If there is no alpha, the color is: red = 5-bit (32 colors), green = 6-bit (64 colors) and blue = 5-bit (32 colors). With 1-bit alpha (simple black or white), the color is: alpha = 1-bit (2 colors), red = 5-bit (32 colors), green = 5-bit (32 colors) and blue = 5-bit (32 colors). With greater than 1-bit alpha (grayscale), the color is: alpha = 4-bit (16 colors), red = 4-bit (16 colors), green = 4-bit (16 colors) and blue = 4-bit (16 colors).

16 bit color is good for strongly contrasting color and alpha that does not compress well.

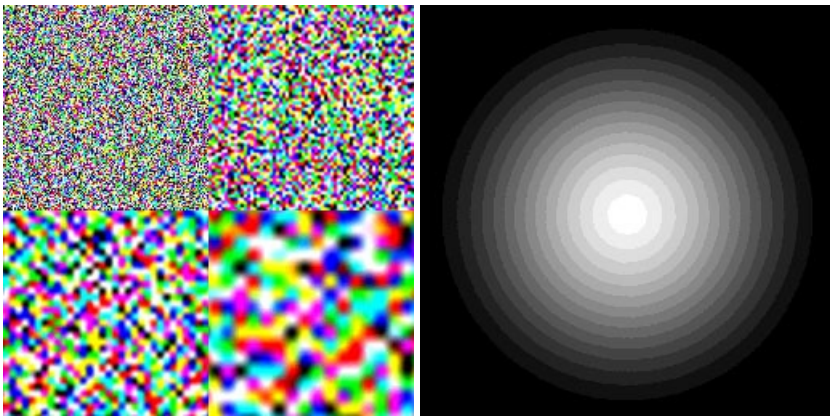


Fig 9. 16-bit color 170kb

32-bit color

Uses 32 bits per pixel, the same as a standard Photoshop file. Note that the alpha channel uses up 8 bits whether you use it for anything or not. Alpha = 8-bit (256 colors), red = 8-bit (256 colors), green = 8-bit (256 colors) and blue = 8-bit (256 colors).

32 bit color is the most expensive setting and should be used sparingly. Used for height maps, which are then internally converted to a special palletized format.

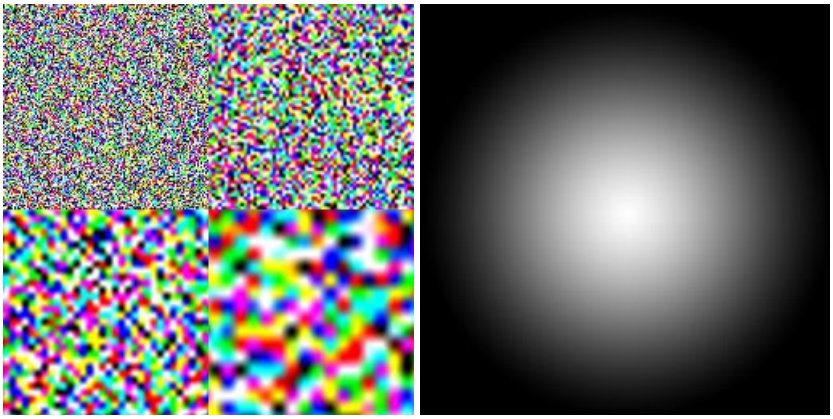


Fig 10. 32-bit color 341kb

Monochrome

Monochrome assumes a gray-scale "color" map with or without alpha. Uses either 8 or 16 bits per pixel, depending on the presence of an alpha channel.

Monochrome is good when you have fine gray-scale information that you need to display. Check out DXT4/5 also, which can produce similar results at a lower cost.

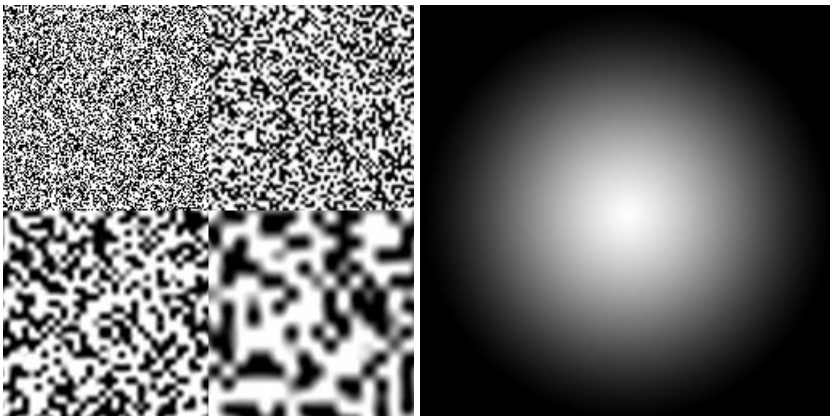


Fig 11. monochrome 170kb

USAGE

Usage determines how mipmaps are generated.

Alpha Blend

Use for alpha-blended textures. Zero alpha pixels are ignored in the downsampling so that the transparent color doesn't bleed (the border between zero and non-zero alpha pixels stays the same).

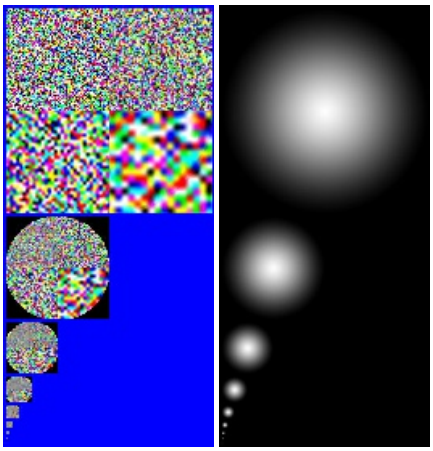


Fig 12. Alpha Blend

Default

Normal downsampling for mipmaps

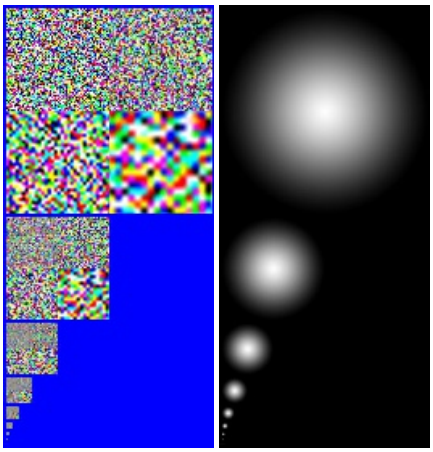


Fig 13. Default

Height Map

Height maps use colors to represent vectors so downsampling has to be handled in a different way.

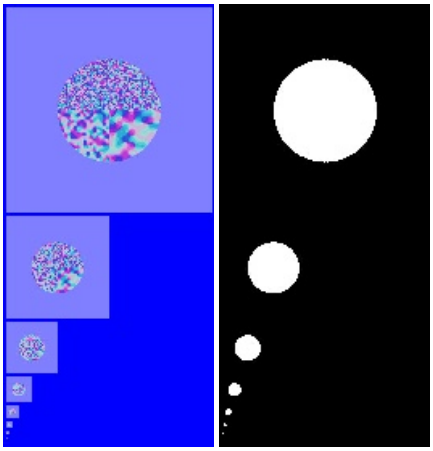


Fig 14. Height Map

**Detail Map

Color fades to gray, set by "detail fade factor" in tag. In this example detail fade factor= .75; if it was set to "1" then the first mipmap would be all gray.

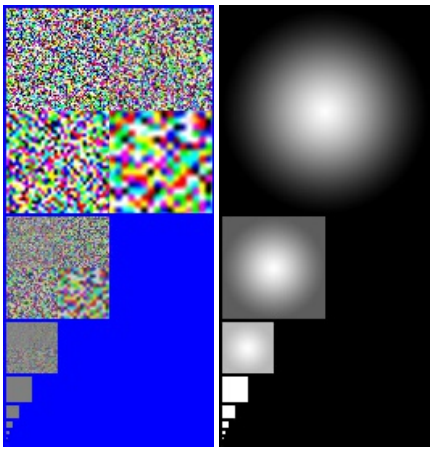


Fig 15. Detail Map

Vector Map

Similar to Height Maps.

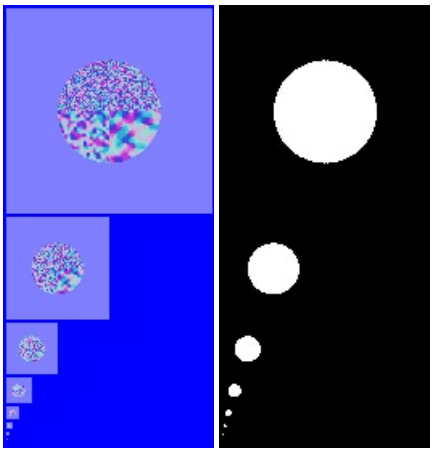


Fig 16. Vector Map

More information on bitmaps can be found in these additional pages.

Bitmap Plates

Import Bitmaps into the Game

Bitmaps - Plates

11/23/2022 • 5 minutes to read

A plate is a number of images assembled in a single .tif file, that when imported creates a single .bitmap file. These can be used to create animated bitmaps or sprites. Animated bitmaps can be used in shaders for animated or random effects, or for animated images in effects. Sprites are used for decorators, effects, lens flares, light volumes and contrails.

BITMAP PLATE BASICS

The importer uses the first three pixels in the upper left corner to set the background color, the border color and the transparent color. The background color is ignored at import, the border color separates the final images and the transparent color is used to change the registration of the images. These three defining colors are a single pixel each.

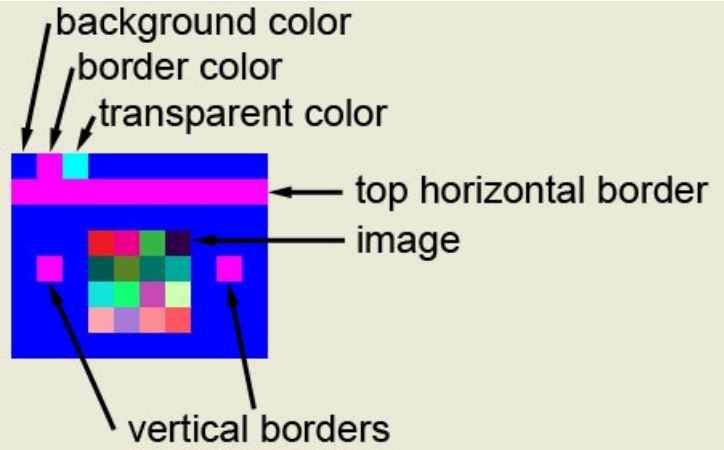


Fig 1. Basic Bitmap Plate Setup.

The background color is ALWAYS Blue (R=0, G=0, B=255). Note that ANY bitmap that has a pure blue pixel in the upper left will be assumed to be a plate by the importer!

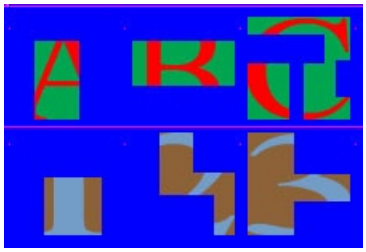
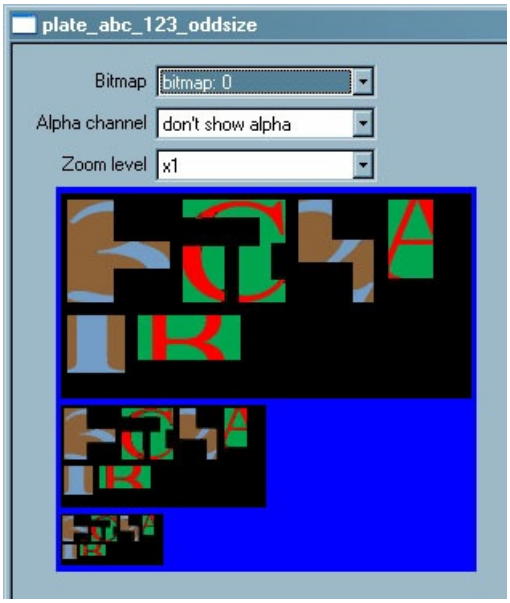
Horizontal borders are one pixel high and must stretch from edge to edge. The top border MUST be the second pixel row. Vertical borders between images must be a single pixel only and surrounded by at least a pixel width of background color.

The registration point for sprites is always the center of the image. The transparent color is used to offset the final image by filling a larger space with a transparent color.

It is very important that you don't allow any variation (aliasing, etc) of these colors. In the background, for example, ANY color other than pure blue (R=0, G=0, B=255) will NOT import as background. This is a common mistake and one you should check for if there are any problems.

A note on odd-size (not power of two dimensions) images:

Sprites can be of any size. At first import, Tool will balk at the odd-size images - but just set the .bitmap type to **sprite** and re-import.

IMPORT	PROCESSED
	
Fig 2. Odd Size Import	Fig 3. Processed import

EXPERT MODE - Changing the Border Width

When processing bitmaps, Tool defaults to creating a 4 pixel border around every plate - 8 pixels between plates. Generally this is fine and helps ensure that mipped plates don't bleed into one another. However there are times when a 8 pixels between plates takes up too much too much room and bumps you up a bitmap size, resulting in wasted texture space. If you want a tighter border to keep the texture size down you do have control over that.

1. You first have to enable Expert Mode in Tool, which is an option under the Edit menu of Guerilla. (BE AWARE THIS CAN CAUSE ALL SORTS OF PROBLEMS - BE CAREFUL!)
2. In the bitmap tag look for the Sprite Spacing string under the ...more sprite processing heading. This is the number of pixels Tool draws around every plate. Notice it defaults to 0, which is equal to a 4 in the eyes of Tool. Change it to 1, 2, or 3 and rerun Tool to process the bitmap with the new setting. This reduces your plate gutter to 2, 4, or 6 pixels.
3. After you are done, disable Expert Mode.

Plates are used to create multiple images in a single .bitmap tag. The Tool command **plate** collects .tif files from a folder and creates a single plate out of them. This plate can be imported into the game as a sequence of bitmaps.

CREATING PLATES USING TOOL

The Tool command **plate** collects .tif files from a folder and creates a single plate out of them. This plate can be imported into the game as a sequence of bitmaps.

The format for running plate is:

```
plate <source-directory>
```

source-directory: Complete path to output tif file, relative to the **directory you are running tool from**. Note that this is different than most other tool commands, which are relative to H2EK\data\ (or your actual working folder).

EXAMPLE

Suppose you have four .tif bitmaps in H2EK\data\test\bitmaps\plate_test



Fig 4. Example images before turning into a Bitmap Plate.

From \H2EK\ you would run:

```
tool plate data\test\bitmaps\plate_test\
```

...which will show you:

```
H2EK>tool plate data\test\bitmaps\plate_test\
found 4 bitmaps...
0: "bands.tif" (128x128)
1: "chip.tif" (128x128)
2: "fall.tif" (128x128)
3: "sky.tif" (128x128)

plate will be #548x#140 with #1 sequences.
successfully created 'data\test\bitmaps\plate_test\.tif'.
```

...and has created H2EK\data\test\bitmaps\plate_test.tif:

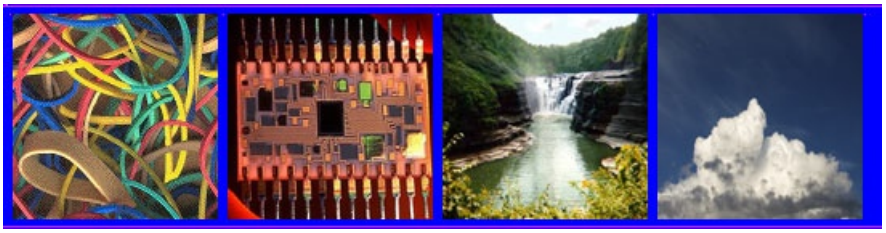


Fig 5. Images now combined into Bitmap Plate.

If you import this into the game using:

```
tool bitmaps test\bitmaps
```

...you will create a single .bitmap tag that contains all four images:

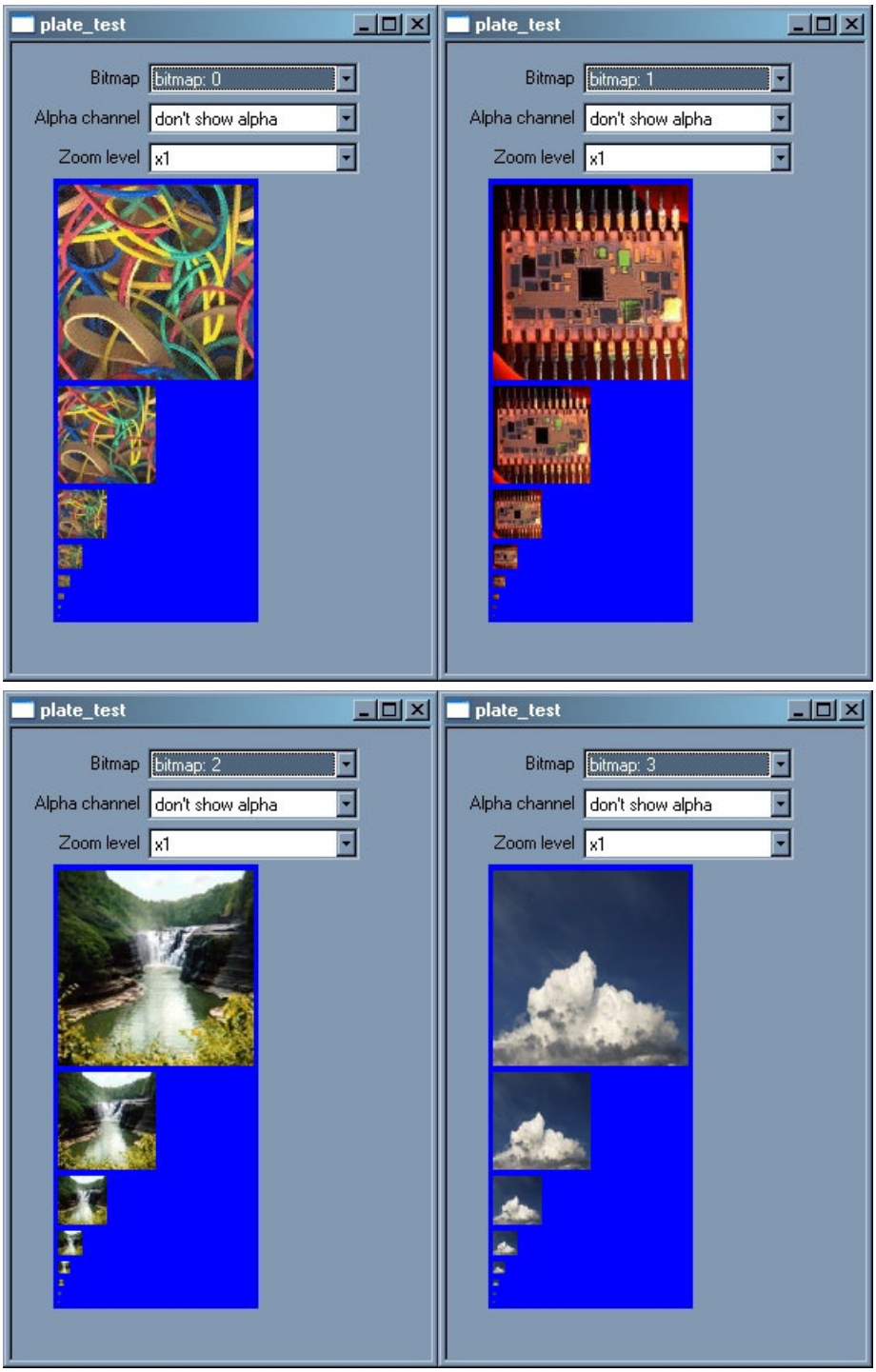


Fig 6. View of the Example Images in Bitmap Tags.

PLATES FOR ANIMATED BITMAPS

Animated bitmaps use the bitmap index to select one of a number of images in a single .bitmap tag.

The images must be in powers of two, though the plate itself can be any size. Animated bitmaps can only have a single row of images.

Here is a simple plate with three images, magnified 300% for clarity:

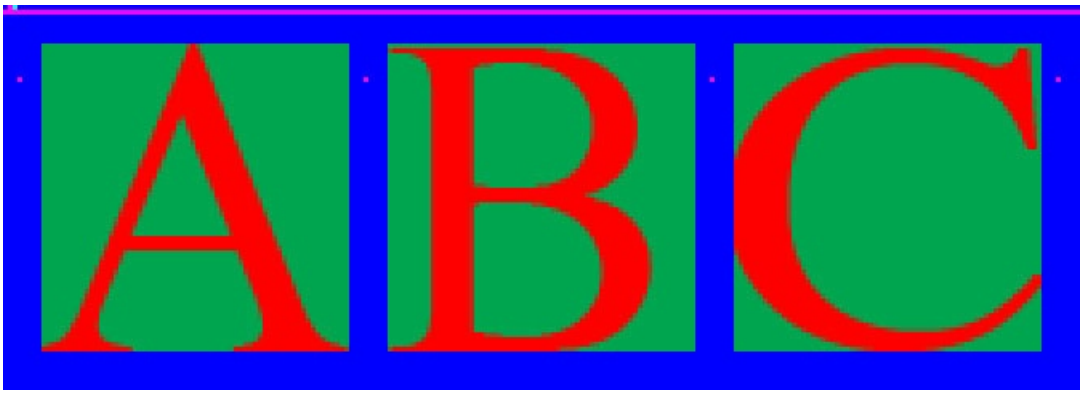


Fig 7. Sample Bitmap Plate.

If this is imported using the regular process for bitmaps, it creates a single .bitmap tag with three indexed images:

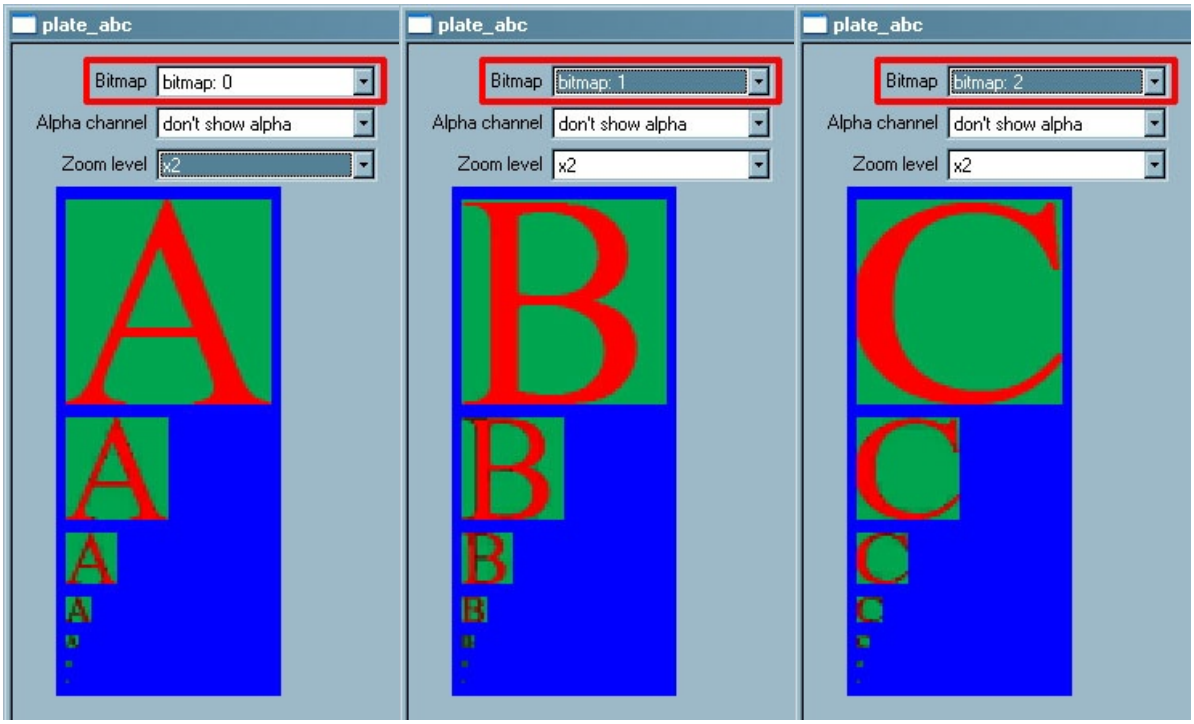


Fig 8.View of the Example Plate in Bitmap Tags.

No special settings are needed in the .bitmap tag.

Animated bitmaps can use different compression settings and can have alpha channels.

PLATES FOR SPRITES

Sprite bitmaps use the **sequence** and **sprite bitmap index** to select one of a number of images in a single .bitmap tag.

The images as well as the plate can be any size. Each row (separated by a horizontal border) will be a unique sequence.

Here is a simple plate with six images (two sequences each with three images), magnified 300% for clarity:

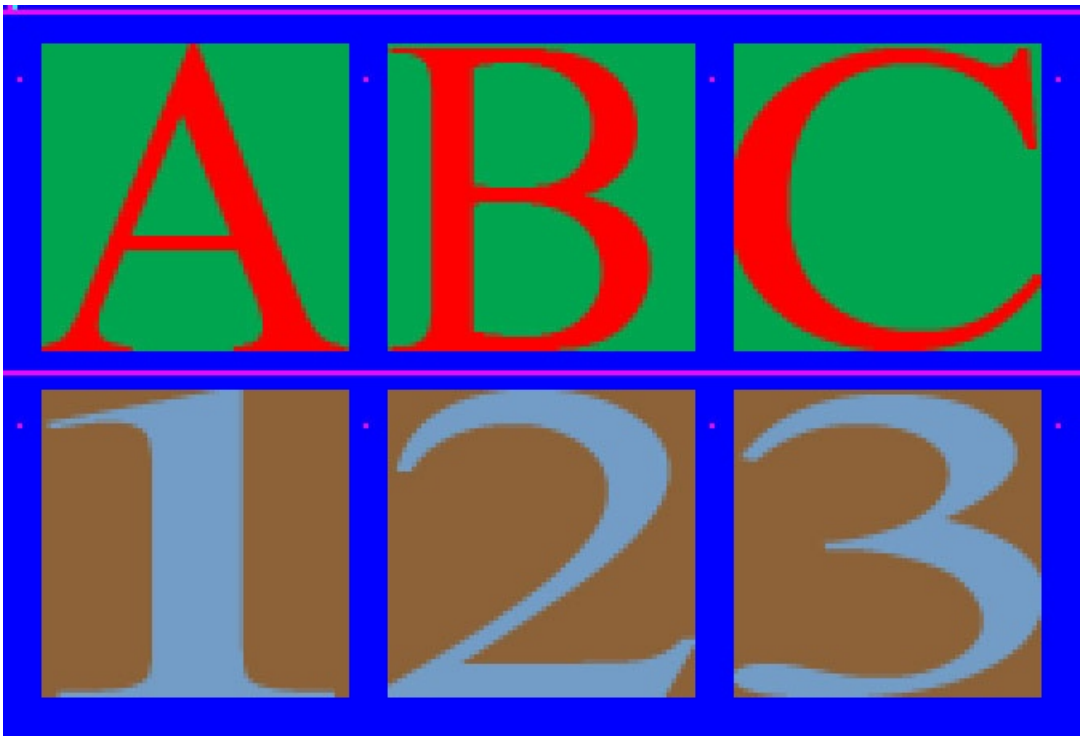


Fig 9. Image sequences example.

If this is imported using the regular process for bitmaps, you'll need to change the bitmap type to **Sprites:**

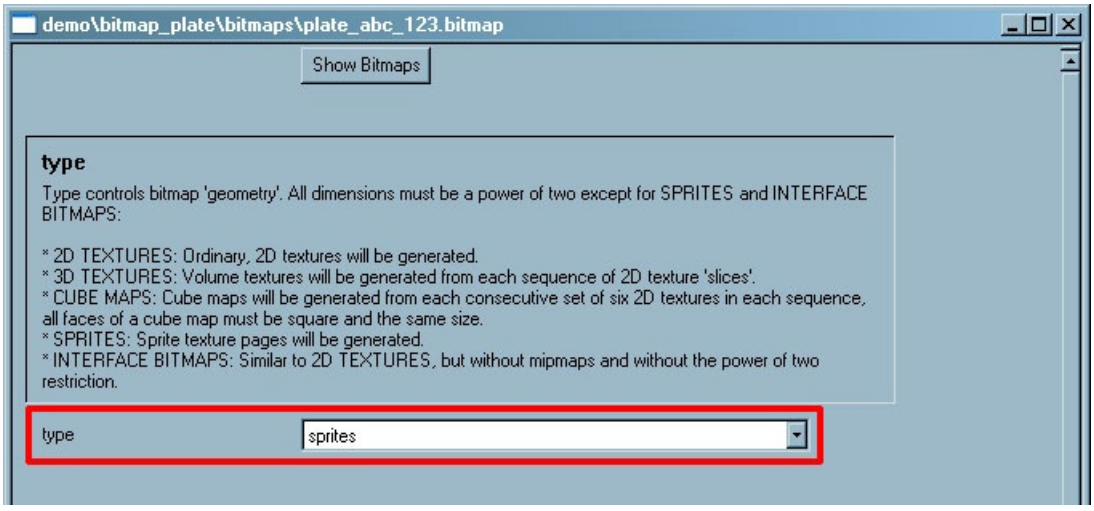


Fig 10. Tag Type set to "sprites".

After re-importing, this will create a single .bitmap tag with six images:

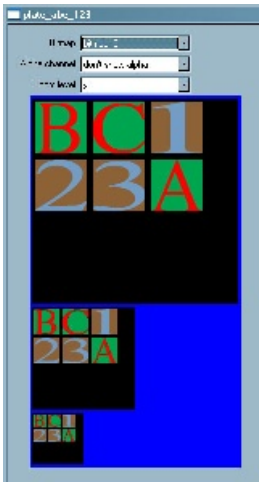


Fig 11. Single .bitmap tag with six images.

NOTE

They will not appear “in order” when viewing the .bitmap tag in Guerilla.

These six images can be addressed by the game engine according to their sequence and sprite bitmap index. You can see this information towards the bottom of the tag:

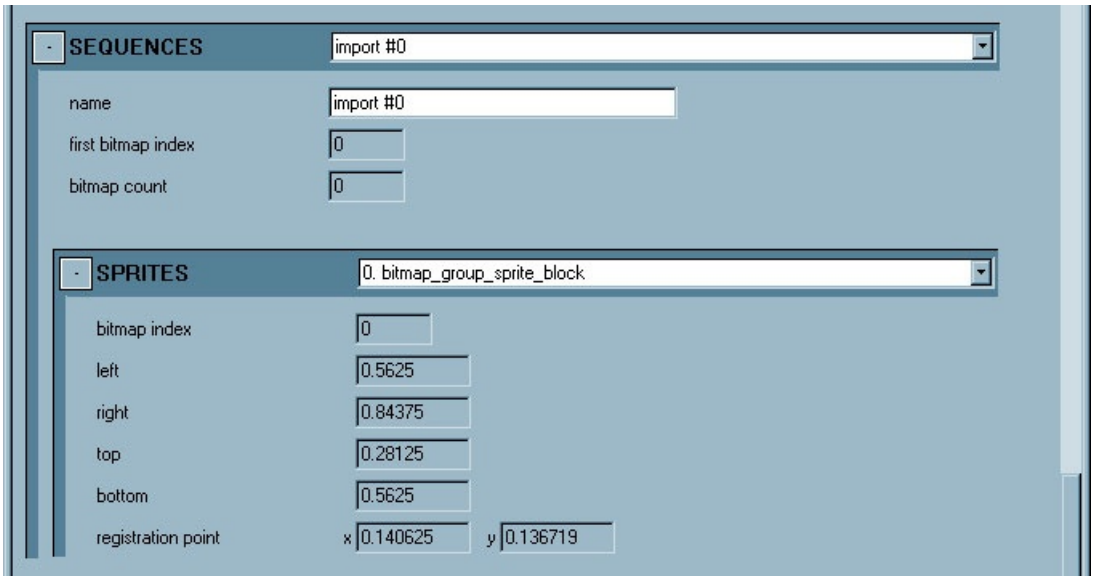


Fig 12. Sprite sequence in Guerilla.

USING TRANSPARENT COLOR FOR SPRITE REGISTRATION

A sprite image will always be placed in the game by its **registration point**, which is always at the center of the image:



Fig 13. The registration point is always in the center of the image.

If you want to change the relative position of the registration point, just add transparent color to center the registration point:

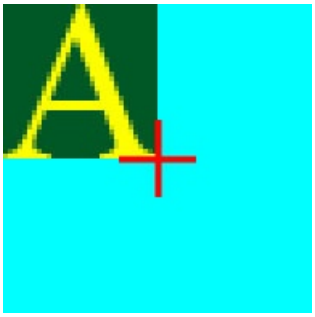


Fig 14. Adding transparency to adjust the registration point.

A plate that looks like this:

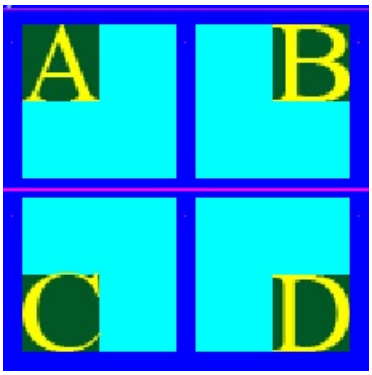


Fig 15. Example plate with the registration points set to the corners of each image.

...will you look like this in the tag:



Fig 16. Example Plate imported into a bitmap tag.

> **NOTE**
>
> The transparent color is not imported - only the registration point is changed.

Bitmaps - Importing Bitmaps Into the Game

11/23/2022 • 2 minutes to read

This page will show you how to import bitmaps into the game using either a command line interface or Guerilla and show you how to take advantage of the various compression schemes and formatting available.

> **IMPORTANT**
> Screenshots are taken from the developer build of Halo 2, some menus will be slightly different in the H2EK.

Importing Bitmaps Using a Command Line Interface

Bitmaps are imported using Tool. You can use the guerilla's interface to run Tool, but it's still running the following steps.

In this example, we're going to import the bitmaps that are in the directory **halo2\data\test\bitmaps**.

Open a command prompt by going start -> run and typing "cmd" in the open window.

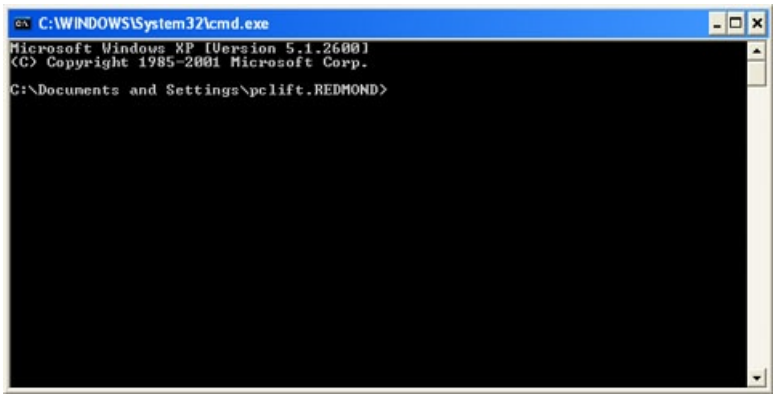


Fig 1. Command Prompt Window.

Navigate to your working game directory by typing "cd c:\halo2" (if your directory is named differently or located on a different drive change this accordingly).

We're going to use the "bitmaps" function of tool.exe (which is in your Halo 2 directory) on all of the bitmaps in the folder **halo2\data\test\bitmaps**.

Type in the command:

```
tool bitmaps test\bitmaps
```

...and hit any key named "enter".

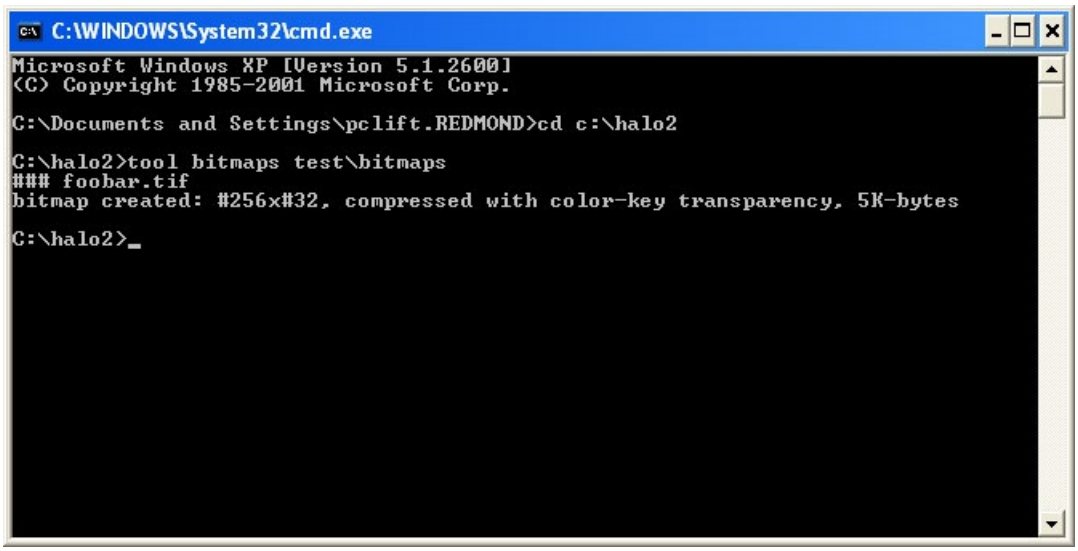


Fig 2. tool bitmaps command.

Tool found one valid bitmap, named foobar.tif, in the folder. It successfully created a bitmap that was 256 pixels wide by 32 pixels high and compressed it with color-key transparency. It created a tag called foobar.bitmap in the halo2\tags\test\bitmaps folder.

Importing Bitmaps Using Guerilla's Interface

In this example, we're going to import the bitmaps that are in the directory **halo2\data\test\bitmaps**.

Open Guerilla.

Under "File", click on "Run Tool" (ctrl+shift+T)

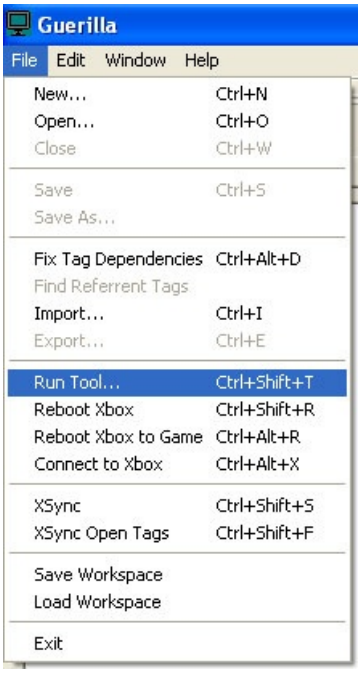


Fig 3. Run Tool option in Guerilla.

A Dialog box will come up. Set the Tool Command to **bitmaps** and source-directory to **test\bitmaps**.

Click on the "OK" button.

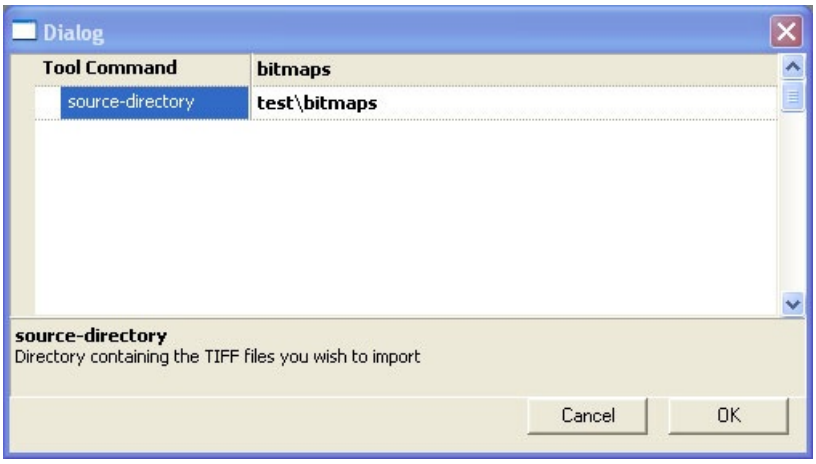


Fig 4. Run Tool dialog window.

You'll see a command prompt appear on your screen and automatically run through the same steps as in the previous example. When it's finished it will leave a Tool Output window which lists what tool actually processed, as well as any errors that it might have encountered.

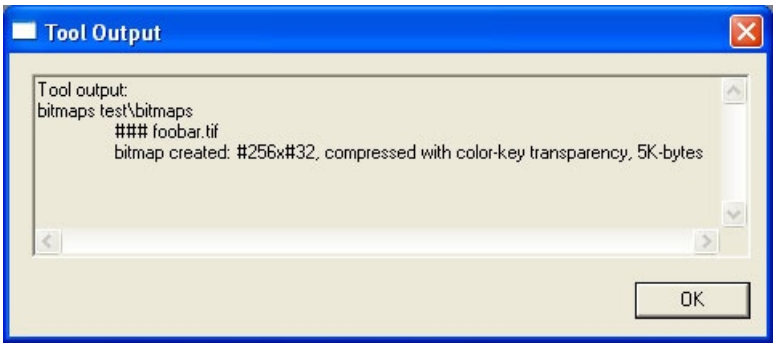


Fig 4. Run Tool output window.

In this case, Tool found one valid bitmap, named foobar.tif, in the folder. It successfully created a bitmap that was 256 pixels wide by 32 pixels high and compressed it with color-key transparency. It created a tag called **foobar.bitmap** in the **halo2\tags\test\bitmaps** folder.

H2 Decorators Overview

11/23/2022 • 5 minutes to read

The decorator system is used to create a large number of (relatively) low-cost objects in an environment, such as grass or rock.

Decorators generally fall into three types: **models**, **decals and quads**. While models use geometry defined in a 3D application, both decals and quads are **sprites** that use a simple pre-defined geometry to display a bitmap. Decals are placed on the surface of BSP geometry while quads can be placed anywhere in a level.

Decorators use a simplified shader system but also save resources by sharing the same shader between decorators. For this reason **you should always use the minimum number of shaders as possible**. All decorators must use .shader_templates in tags\shaders\shader_templates\decorators\

SAPIEN NOTE

To reset the placement and lighting of decorators in Sapien, use the console command

```
decorator_rebuild_all
```

DECORATOR CLASSES

There are six types of decorators: **model, floating decal, projected decal, screen facing quad, axis rotating quad**, and **cross quad**.

Model

Model is the only class of decorators that you can build geometry for. Objects are created in your 3D application as a bulk file, and exported as a .jmi file. Note that all decorator models are two-sided. No parsed materials.

- Source: .jmi model file (bulk file, can have mutiple elements)
- Import: use tool: import-decorator-set
- Creates .decorator_set
- Shader: use decorator_tex.shader_template only.
- Lighting: lit by lightmaps, not lit by dynamic lights
- LOD: smallify w/distance

> **NOTE**
> No damage states, does **not** accept decals

Floating Decal

A floating decal is a sprite that is placed on the surface of BSP geometry in Sapien. Source is from bitmap only - there is no user-defined geometry. The default size of the final sprite in game is 256 pixels per world unit.

- Source: .tif bitmap file (sequenced sprite plate)
- Import: use tool: bitmaps (set .bitmap = sprite, sprite_usage for blend function)
- Creates: .bitmap tag (.decorator_set must be created or edited seperately)

- Shader: use decorator_decal_paint.shader_template (alpha blended), decorator_decal_paint_additive.shader_template for additive or decorator_decal_paint_multiply.shader_template for multiplicative
- Lighting: lit by lightmaps from underlying object (uses ovelay), lit by dynamic lighting from underlying object (uses ovelay)
- LOD: fade w/distance

> **NOTE**
> Will accept decals

Projected Decal

A projected decal is a sprite that is projected on to a surface of BSP geometry in Sapien. It has the same attributes as a floating decal, except it is projected (smeared) onto the surface so that it will follow the contours of the underlying geometry. Source is from bitmap only - there is no user-defined geometry. The default size of the final sprite in game is 256 pixels per world unit.

- Source: .tif bitmap file (sequenced sprite plate)
- Import: use tool: bitmaps (set .bitmap = sprite, sprite_usage for blend function)
- Creates: .bitmap tag (.decorator_set must be created or edited seperately)
- Shader: use decorator_decal_paint.shader_template (alpha blended), decorator_decal_paint_additive.shader_template for additive or decorator_decal_paint_multiply.shader_template for multiplicative
- Lighting: lit by lightmaps from underlying object (uses ovelay), lit by dynamic lighting from underlying object (uses ovelay)
- LOD: fade w/distance

> **NOTE**
> Will accept decals

Screen Facing Quad

A Screen Facing Quad is a free-standing sprite that always faces the camera.

- Source: .tif bitmap file (sequenced sprite plate)
- Import: use tool: bitmaps (set .bitmap = sprite, sprite_usage for blend function)
- Creates: .bitmap tag (.decorator_set must be created or edited seperately)
- Shader: use decorator_tex.shader_template only.
- Lighting: lit by lightmaps, not lit by dynamic lights
- LOD: smallify w/distance

> **NOTE**
> No damage states, does not accept decals

Axis Rotating Quad

An Axis Rotating Quad is a free-standing sprite with one axis always along normal of surface that it is placed on with the other two always facing the camera.

- Source: .tif bitmap file (sequenced sprite plate)
- Import: use tool: bitmaps (set .bitmap = sprite, sprite_usage for blend function)
- Creates: .bitmap tag (.decorator_set must be created or edited seperately)
- Shader: use decorator_tex.shader_template only.
- Lighting: lit by lightmaps, not lit by dynamic lights
- LOD: smallify w/distance

> **NOTE**
> No damage states, does not accept decals

Cross Quad

A Cross Quad is a free-standing sprite comprised of two cards perpendicular to each other. They are fixed so do not move with the camera.

- Source: .tif bitmap file (sequenced sprite plate)
- Import: use tool: bitmaps (set .bitmap = sprite, sprite_usage for blend function)
- Creates: .bitmap tag (.decorator_set must be created or edited seperately)
- Shader: use decorator_tex.shader_template only.
- Lighting: lit by lightmaps, not lit by dynamic lights
- LOD: smallify w/distance

> **NOTE**
> No damage states, does not accept decals

.decorator_set TAG

.decorator_set tag is created when importing a decorator model using import-decorator-set, or they can be created in Guerilla by selecting **File - New** and setting Group equal **.decorator_set**.

SHADERS

Entries for all shaders used in the decorator set.

shader: link to .shader tag(s).

lighting min scale: range: 0-2 (0 defaults to .4) - range for how 'directional' light is, with 0 = ambient/diffuse and 2 = very directional (sharp shadows).

lighting max scale: range: 0-2 (0 defaults to 2).

CLASSES

Entries for classes used in the decorator set. You can have multiple classes per set.

name: name that will be shown in Sapien.

type: model, floating decal, projected decal, screen facing quad, axis rotating quad, or cross quad

scale: controls size of entire class with one being equal to what is authored.

PERMUTATIONS

Must have an entry for each permutation.

name: name that will be shown in Sapien.

shader: pick from shaders defined above.

flags:

- **align to normal**: will align with normal of face decorator is placed on.
- **only on ground**: will only appear on horizontal surfaces but not on vertical.
- **upright**: vertical axis locked.

fade distance:

- **close**: start fade = 53' finish fade = 80'
- **medium**: start fade = 107' finish fade = 160'
- **far**: start fade = 160' finish fade = 240'

index: set index for specific decorator to be used in this permutation. For models, this will be the index as listed in the MODELS block (see below). For sprites (quads and decals) this will be the .bitmap tag sprite index.

distribution weight: weight for this permutation when put down in a ramdom distribution in Sapien. This number is simply compared with the other entries in the permutation. Zero will NOT appear in random distribution.

scale: controls size of this permutation with one being equal to what is set in class scale (see above).

tint 1: permutation will be tinted (multiplied) by this color range with white having no effect.

tint 2: see above.

base map tint percentage: samples base map of decorator placed upon - straight multiply! try this with grayscale only (no color) ? not for decals?

lightmap tint percentage: models and sprites ONLY not for decals samples lightmap double-multiply so gray lightmap has no effect with zero equals self-illuminated

wind scale: does NOT work.

THERE IS NO USER-EDITABLE INFORMATION BELOW THIS POINT

MODELS

lists the [index number]: [model name] (as named in the origin node in the bulk file)

> **NOTE**
> That the **index** is what gets referenced in the PERMUTATION, not the **name**

model name: as named in the origin node in the bulk file

index start:

index count:

RAW VERTICES

position:

normal:

tangent:

binormal:

texcoord:

INDICES

index:

CACHED DATA

BLOCK INFO

block offset:

block size:

section data size:

resource data size:

RESOURCES

owner tag section offset:

Additional information on Decorators can be found in the following pages:

Floating Decals

Models

Screen Facing Quads

H2 Decorators - Floating Decals

11/23/2022 • 3 minutes to read

A **floating decal** is a sprite that is placed on the surface of BSP geometry in Sapien. Source is from bitmap only - there is no user-defined geometry. The default size of the final sprite in game is 256 pixels per world unit.

- Source: .tif bitmap file (sequenced sprite plate)
- Import: use tool: bitmaps (set .bitmap = sprite, sprite_usage for blend function)
- Creates: .bitmap tag (.decorator_set must be created or edited seperately)
- Shader: use
 - decorator_decal_paint.shader_template for alpha blended effects
 - decorator_decal_paint_additive.shader_template for additive
 - decorator_decal_paint_multiply.shader_template for multiplicative
- Lighting: lit by lightmaps from underlying object (uses ovelay), lit by dynamic lighting from underlying object (uses ovelay)
- LOD: fade w/distance

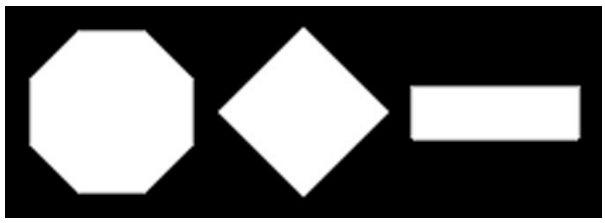
> **NOTE**
> Will accept decals

SETTING UP THE SPRITE PLATE

A plate is a specially set up .tif file which creates a single .bitmap tag that can contain multiple images. These images are in a single sprite sequence and can be accessed from the .decorator_set tag by using the **sprite index**.

Sprites can be any size - dimensions do not have to be in powers of two. Remember, the default size of the final sprite in game is 256 pixels per world unit.

For this example, we're going to use a plate which includes three images and an associated alpha (click image to see it magnified, for clarity):

DIFFUSE	ALPHA
	

Note that the blue background is essentially one-bit alpha, plus there is alpha blending in the alpha channel.

The .tif file can be imported as a .bitmap tag using the regular Tool or Guerilla process.

EDIT THE .BITMAP TAG IN GUERILLA

Once the .bitmap tag has been created, open the tag in Guerilla and set **type = "sprites"**. Re-import, and the .bitmap tag will look like this:

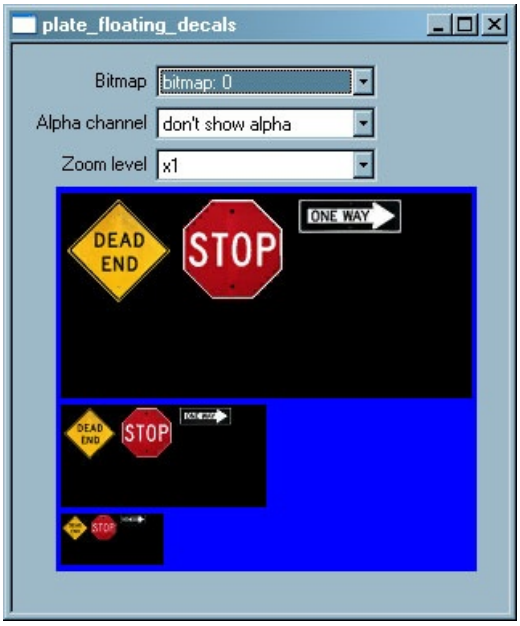


Fig 1. .bitmap Tag in Guerilla.

You can view the **sprite index** near the bottom of the .bitmap tag. The only way to relate the index to the image is to count them - there is no visual feedback for which one is which.

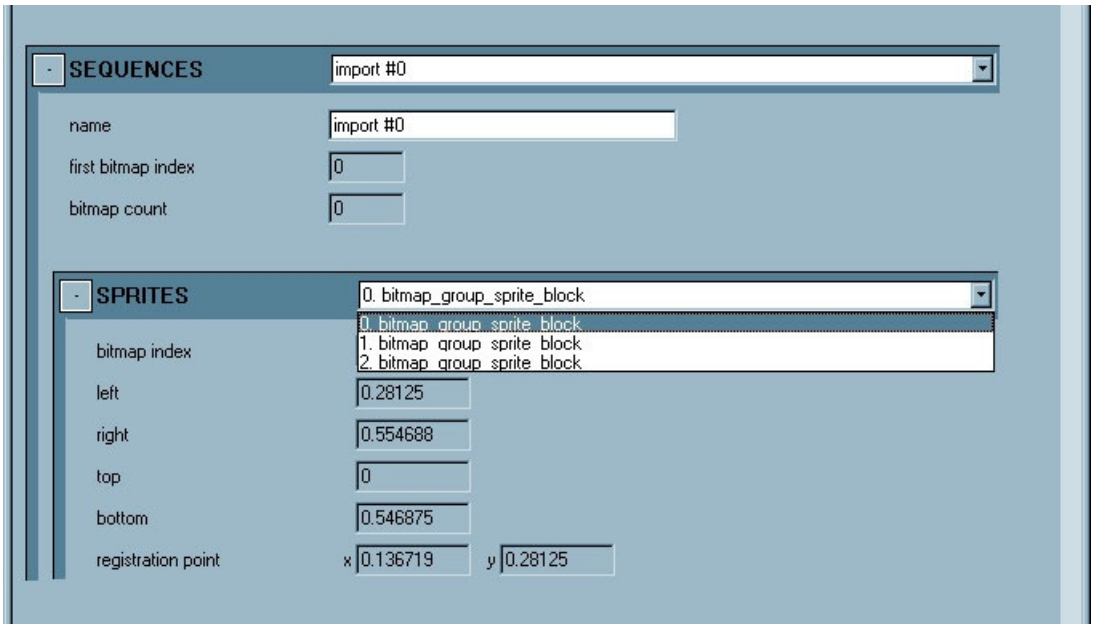


Fig 2. The Sprite Index in the .bitmap Tag.

In this case, the DEAD END sign is index #0, the STOP sign is index #1 and the ONE WAY sign is index #2.

> **NOTE**
> This is **NOT** the order of the original plate!

SETTING UP THE .SHADER TAG

To create the .shader tag: in Guerilla select **File -> New** and set **Group = "shader'**. This will create an empty

.shader tag.

For decorator templates, use only those tags in shaders\shader_templates\decorators. For floating decals, use:

- decorator_decal_paint.shader_template for alpha blended effects
- decorator_decal_paint_additive.shader_template for additive
- decorator_decal_paint_multiply.shader_template for multiplicative

For this case, we'll use **decorator_decal_paint**. Once you have set up the .shader tag, add the reference to the .bitmap tag set up in the previous step. Then it will look like this:

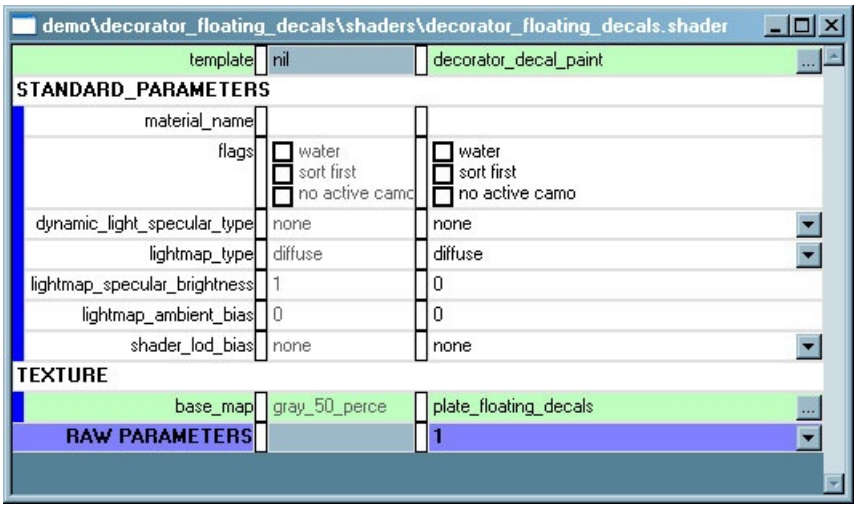


Fig 3. Shader tag with the decorator_decal_paint Template.

> **NOTE**
> None of the standard parameters apply to decorators.

SETTING UP THE .DECORATOR_SET TAG

To create the .decorator_set tag: in Guerilla select **File -> New** and set **Group = "decorator_set'**. This will create an empty .decorator_set tag.

SHADERS BLOCK

In the SHADERS block, choose **Add** and then reference the .shader tag.

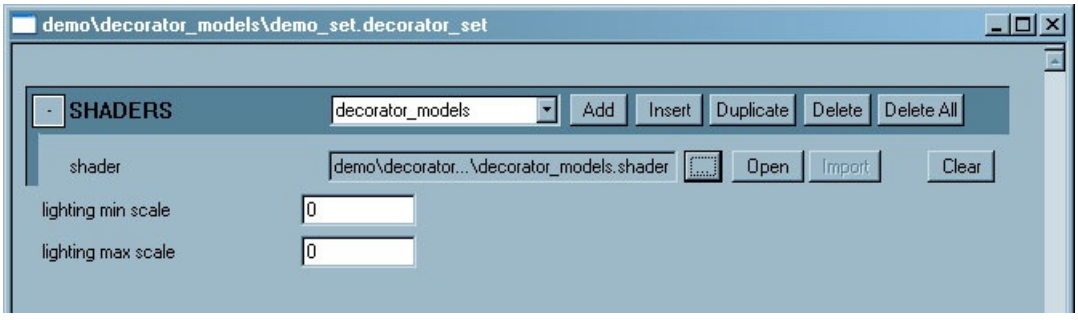


Fig 4. Shaders Block.

Next, you'll need to add at least one **decorator class**. Decorator sets can contain multiple classes but for this example we'll make just one class: **floating decal**.

CLASSES BLOCK

In the CLASSES block, choose **Add**. The **name** field is what this set of decorator models will be shown as in

Sapien.

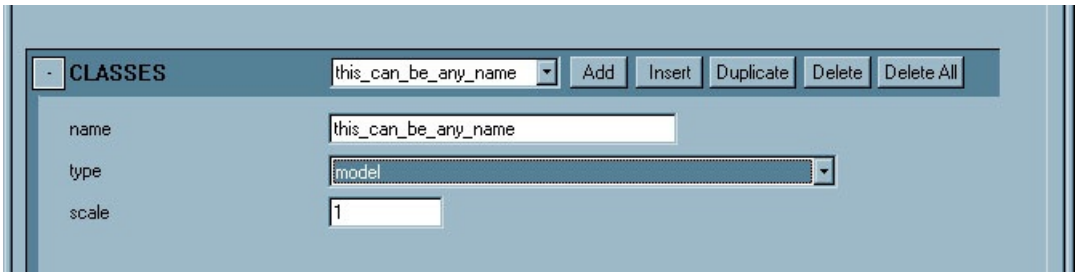


Fig 5. Classes Block.

Next we'll add one entry in the PERMUTATIONS block for each sign. Let's start with the **DEAD END** image, which is index # 0.

PERMUTATIONS BLOCK

In the PERMUTATIONS block, choose **Add**. The **name** field is what this particular instance will show as in Sapien. Note that the flags "align to normal" and "upright" do not apply to floating decals. The **shader** field will allow you to pick any shader defined at the top of the tag in the SHADERS block. For this example, **dead_end** will fade at a close distance. It is referenced by index #0. It has a distribution weight = 1; this will be compared to the weight of the other permutations when we are putting down the **foo** decorator set in a random application.

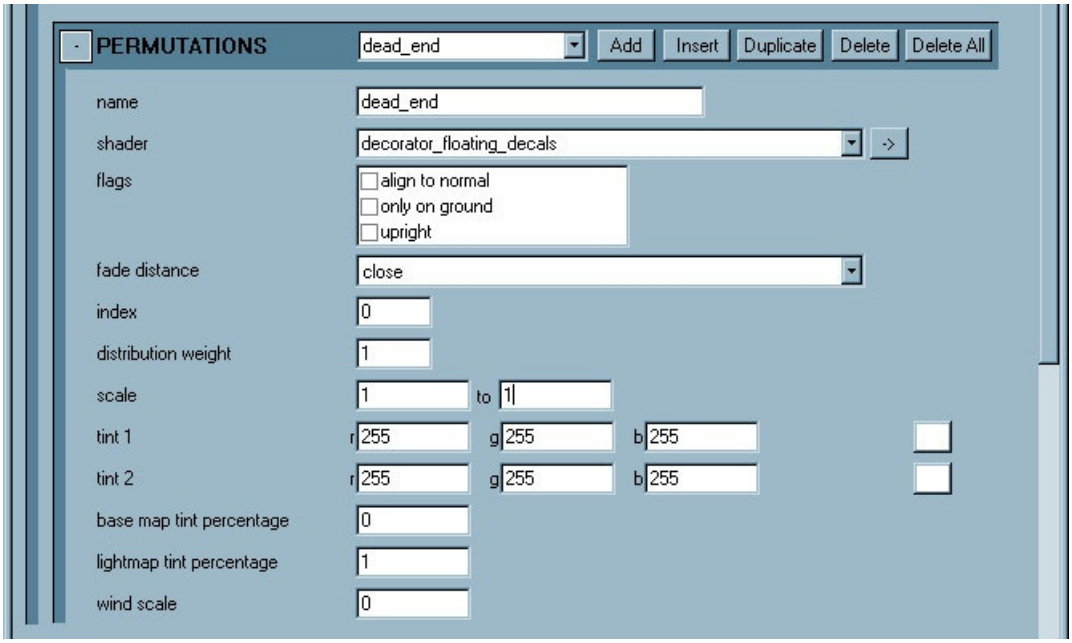


Fig 6. Permutations Block.

Repeat this process for **STOP** (index #1) and **ONE WAY** (index #2). Remember that you can use the "Duplicate" button to make this process easier.

READY FOR PLACEMENT!

This decorator set is now ready to be placed in Sapien.



Fig 7. Decals visible in Sapien.

H2 Decorators - Models

11/23/2022 • 4 minutes to read

Decorator Models are a class of **Decorators**.

Model is the only class of decorators that you can build geometry for. Objects are created in a 3D application as a bulk file, and exported as a .jmi file. Note that all decorator models are two-sided. No parsed materials.

- Source: .jmi model file (bulk file, can have mutiple elements)
- Import: use tool: import-decorator-set
- Creates .decorator_set
- Shader: use decorator_tex.shader_template only. This is an alpha tested shader.
- Lighting: lit by lightmaps, not lit by dynamic lights
- LOD: smallify w/distance

> **NOTE**
> No damage states, does not accept decals

SETTING UP THE DECORATOR MODELS

Decorator Models in the same set are built in a single bulk file. The bulk file will contain render models only - no collision or physics models.

Here is a example single decorator object set up:

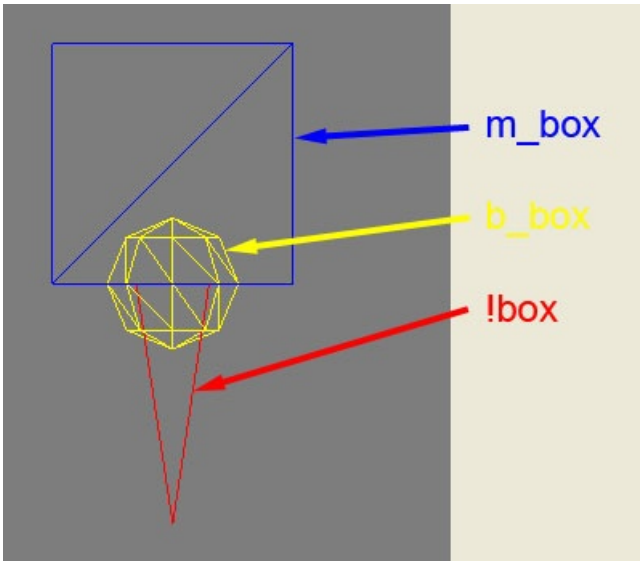


Fig 1. A single decorator object set up.

The **mesh** is a box named "m_box", the **frame node** is a sphere named "b_box" and the the **origin node** is a pyramid named "!box"

Make sure that mesh has UV coordinates and a proper shader applied.

If we want to have three objects in our decorator set - a box, an L extrustion and an oil tank - it might look like

this:

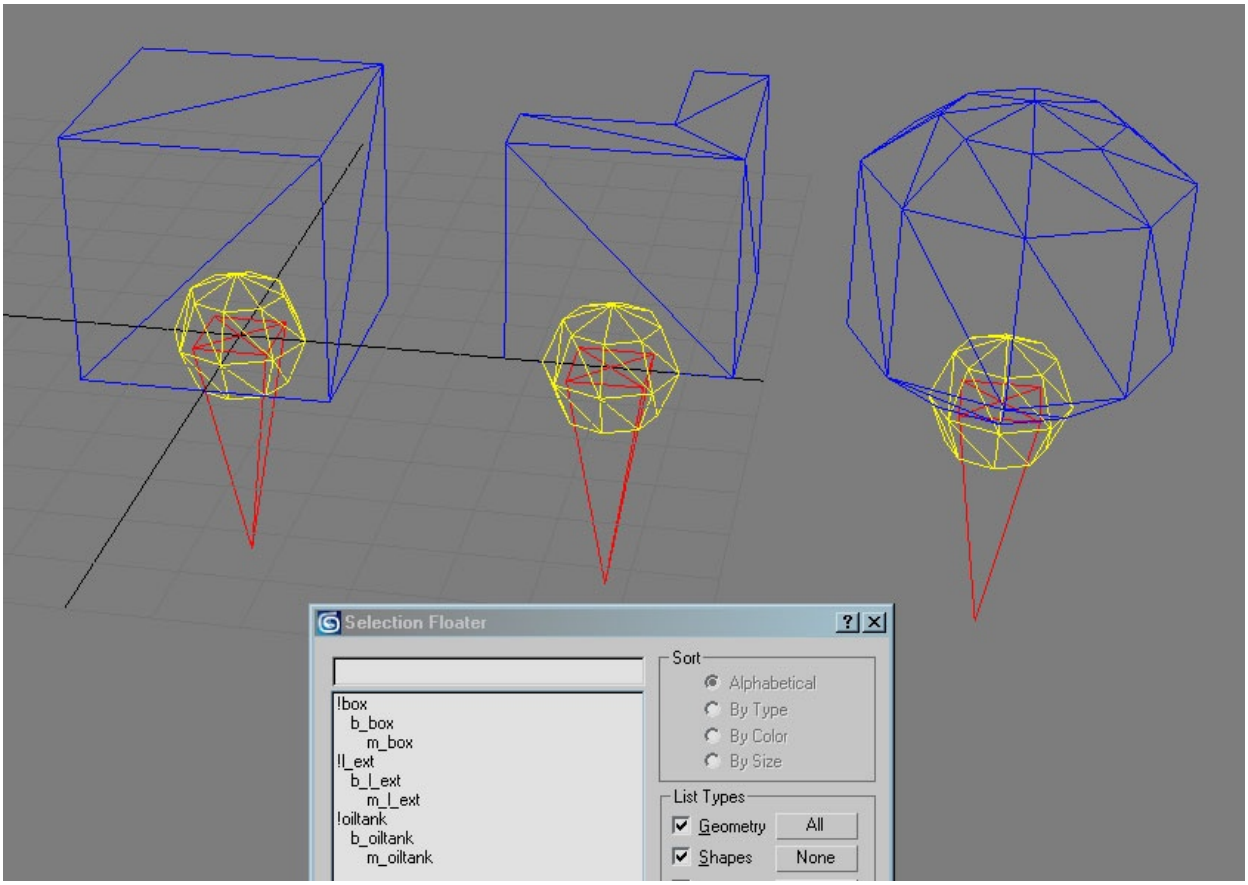


Fig 2. Multiple decorator objects set up.

From here you would export a **.jmi** file into the proper directory.

If the .jmi is exported into a directory called "decorator_models" it will create subdirectories like this:

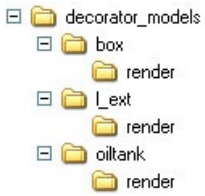


Fig 3. decorator_models directory.

> **NOTE**
> Subdirectories are named after the origin nodes. Each of these subdirectories contains a folder called "render". Each of those render folders contains a .jms file (again named after the corresponding origin node).

IMPORTING THE DECORATOR MODELS

After you have exported the .jmi file (which should have created the correct subfolders and .jms files), you can import this by running Tool from inside Guerilla.

The Tool Command is **import_decorator_set**.

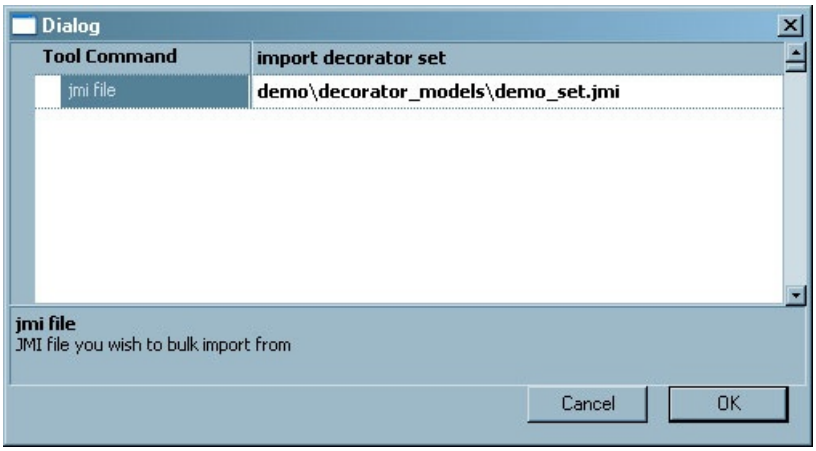


Fig 4. import_decorator_set command in Guerrila.

When you run Tool, you'll get an output window:

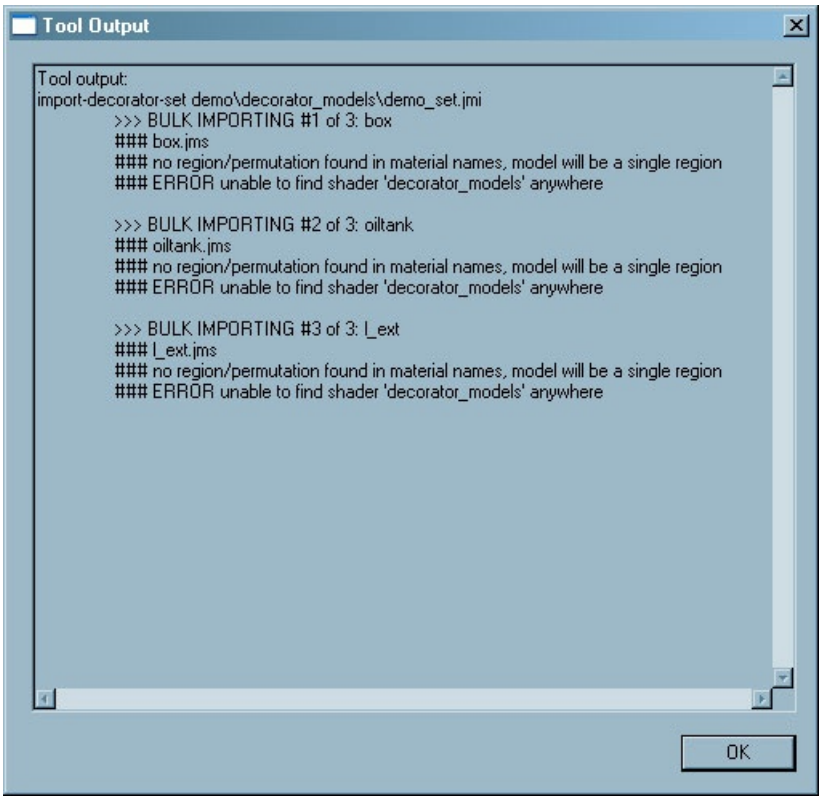


Fig 5. import_decorator_set command tool output.

Note that you will be attaching the shader separately - it doesn't need to be associated with the model at import.

SETTING UP THE .DECORATOR_SET TAG IN GUERILLA

When you imported the .jmi file using **import_decorator_set** it will create a **.decorator_set** tag in the analogous hierarchy as the .jmi file, so:

```
\H2EK\data\demo\decorator_models\demo_set.jmi
```

...will create:

```
\H2EK\tags\demo\decorator_models\demo_set.decorator_set
```

Initially, the tag will only contain the model data:

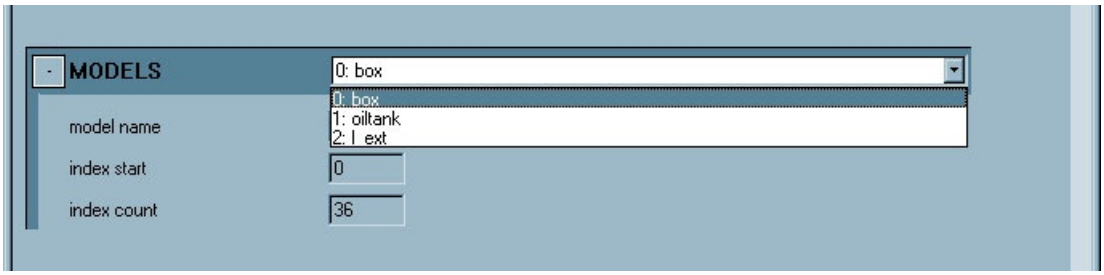


Fig 6. Models Data section of the decorators_set tag.

CREATING THE SHADER

First, you'll need to create the shader to be used by the decorator models. Create a new .shader tag in Guerilla by choosing file -> new and setting Group to shader. Select a template for the .shader by choosing the "..." icon. Navigate into the decorators folder and choose decorator_tex.shader_template. This is the ONLY template you should use with decorator models.

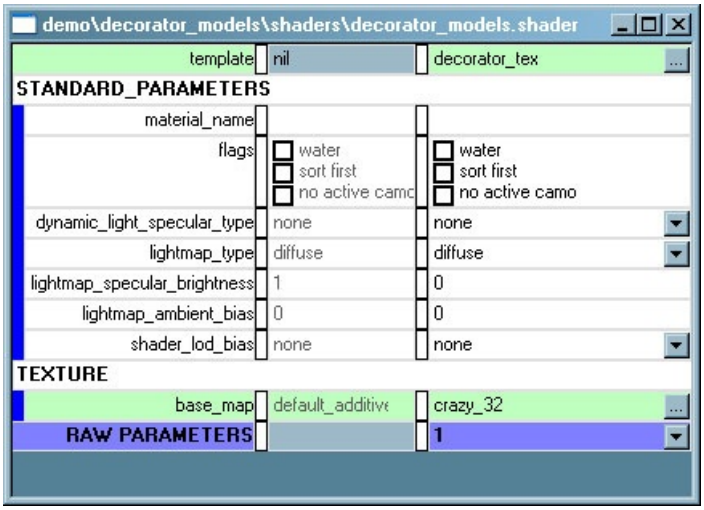


Fig 7. New shader with the decorator_tex.shader_template.

> **NOTE**
>
> None of the standard parameters apply to decorators.

Reference the base map you want to use with the shader.

SHADERS BLOCK

Now that you have a .shader tag you can reference it from the .decorator_set tag. In the SHADERS block, choose **Add** and then reference the new .shader tag. Note that while the .decorator_set can reference multiple shaders, you'll want to use as few as possible.

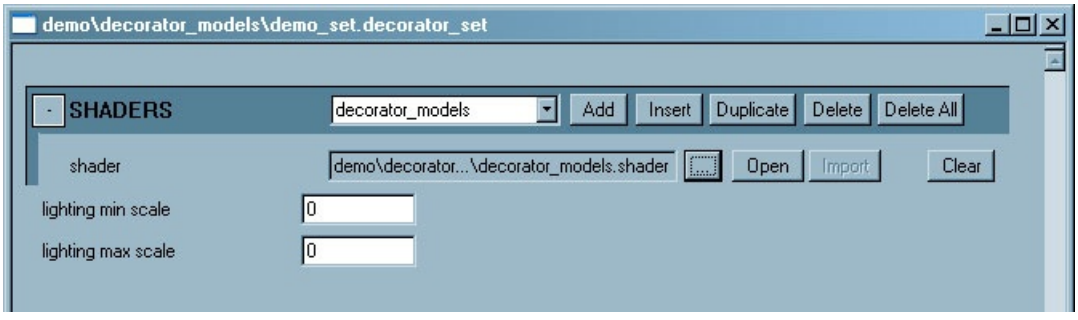


Fig 8. Adding the new shader to the shaders block of the .decorator_set tag.

Next, you'll need to add at least one **decorator class**. Decorator sets can contain multiple classes but for this

example we'll make just one class: **models**.

CLASSES BLOCK

In the CLASSES block, choose **Add**. The **name** field is what this set of decorator models will be shown as in Sapien. The **this_can_be_any_name** set could contain a single model, but for this example we will put all three models in the same set. This will allow all three to be randomly distributed at the same time.

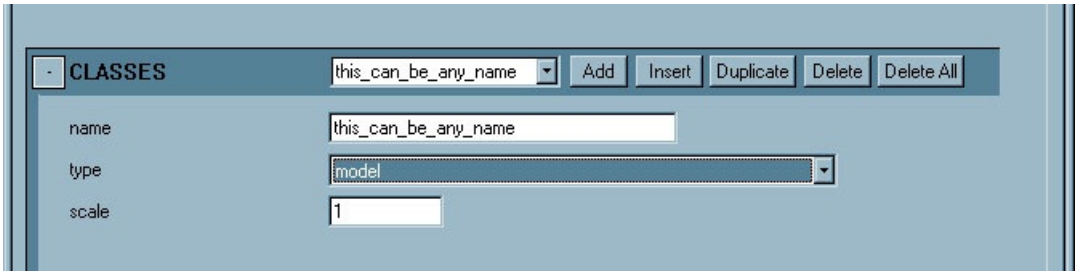


Fig 9. Add a new Class in the Classes block.

Next we'll add one entry in the PERMUTATIONS block for each model. Let's start with the **box** model, which is index # 0.

PERMUTATIONS BLOCK

In the PERMUTATIONS block, choose **Add**. The **name** field is what this particular model will show as in Sapien. The **shader** field will allow you to pick any shader defined at the top of the tag in the SHADERS block. For this example, **sample_box** will only appear on the ground and will fade at a close distance. The **index** for the box can be found in the MODELS block - it's the number that's important - in this case it reads "0:box", so the index for the box model is 0.

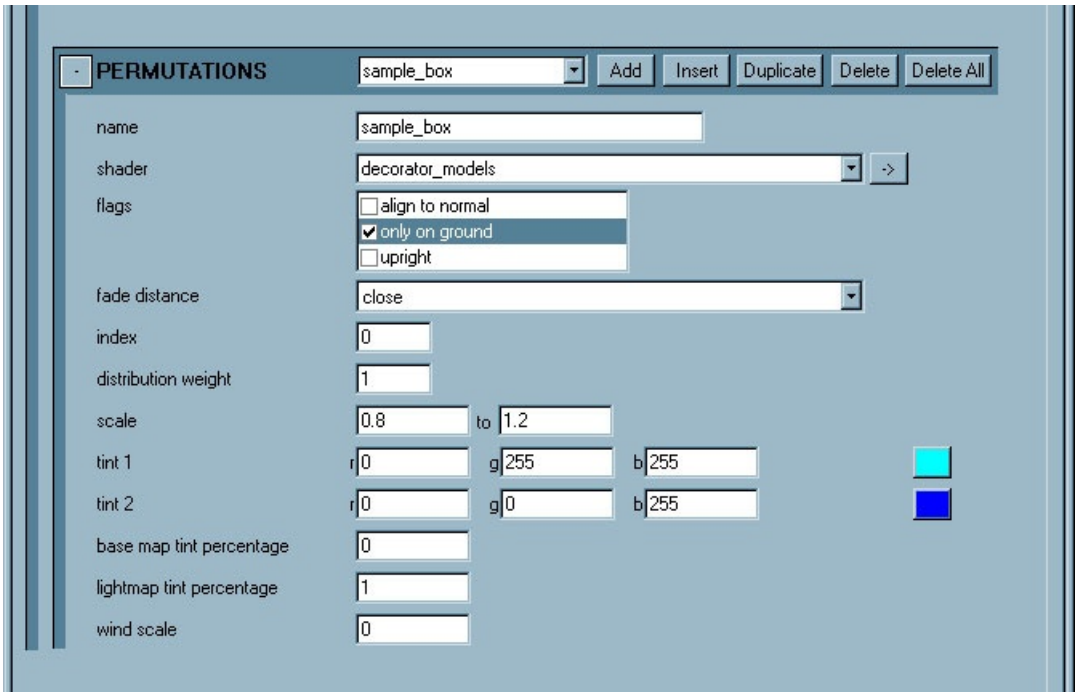


Fig 10. Permutations block in the decorator_set tag.

Repeat this process for the **l_ext** and the **oil_tank**. Remember you can use the **duplicate** function.

READY TO PLACE!

This .decorator_set is ready to be placed in Sapien.

H2 Decorators - Screen Facing Quads

11/23/2022 • 3 minutes to read

A **Screen Facing Quad** is a free-standing sprite that always faces the camera.

- Source: .tif bitmap file (sequenced sprite plate)
- Import: use tool: bitmaps (set .bitmap = sprite, sprite_usage for blend function)
- Creates: .bitmap tag (.decorator_set must be created or edited seperately)
- Shader: use decorator_tex.shader_template only. This is an alpha tested shader.
- Lighting: lit by lightmaps, not lit by dynamic lights
- LOD: smallify w/distance

> **NOTE**
> No damage states, does not accept decals

SETTING UP THE SPRITE PLATE

A plate is a specially set up .tif file which creates a single .bitmap tag that can contain multiple images. These images are in a single sprite sequence and can be accessed from the .decorator_set tag by using the **sprite index**.

Sprites can be any size - dimensions do not have to be in powers of two. Remember, the default size of the final sprite in game is 256 pixels per world unit.

For this example, we're going to use a plate which includes three images (magnified 300% for clarity):

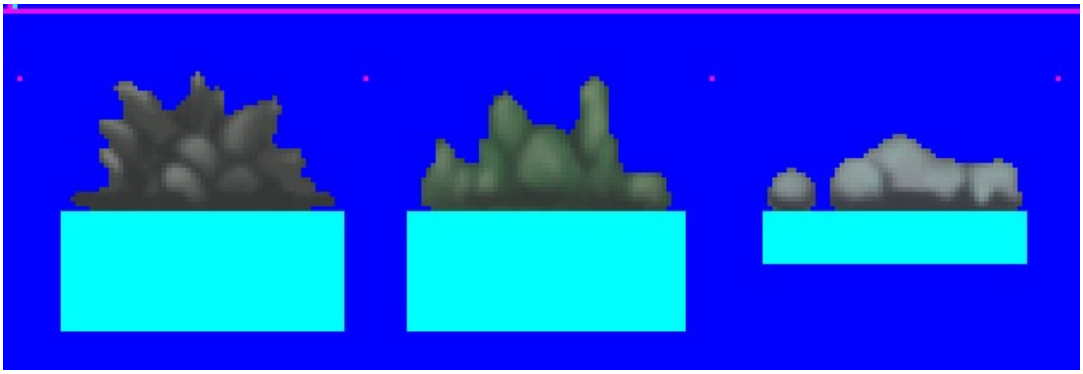


Fig 1. Example bitmap plate.

> **NOTE**
> The blue background is essentially one-bit alpha. You can use this for your alpha channel **IF** you use DXT2/3 compression. This is a bug - it should work with DXT1 (WAC!). You can use DXT1 but you must specify an alpha channel in your .tif file. Note the use of transparent registration color.

The .tif file can be imported using the regular process for .bitmap tags. Once the .bitmap tag has been created, open the tag and set **type = "sprites"**. Re-import, and the .bitmap tag will look like this:

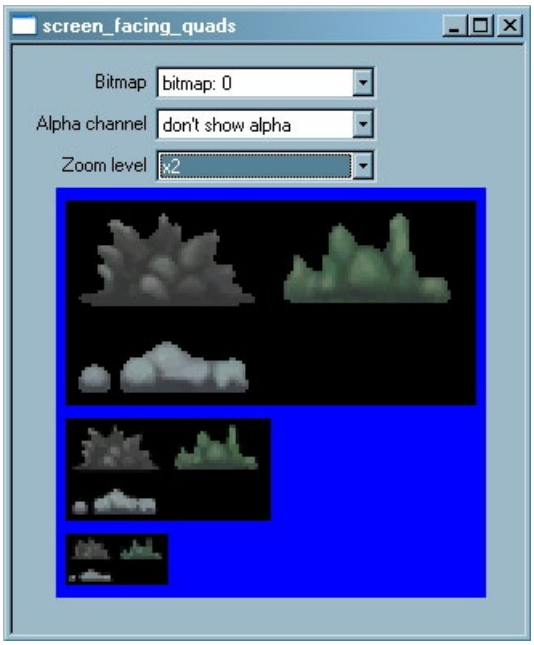


Fig 2. Example imported bitmap tag.

You can view the **sprite index** near the bottom of the .bitmap tag. The only way to relate the index to the image is to count them - there is no visual feedback for which one is which.



Fig 3. Sprite index in the new bitmap tag.

In this case, the sprite index is in the same order as the original .tif plate. This is not always true!

SETTING UP THE .SHADER TAG

To create the .shader tag: in Guerilla select **File -> New** and set **Group = "shader'**. This will create an empty .shader tag.

For decorator templates, use only those tags in *shaders\shader_templates\decorators*. For floating decals, use *decorator_tex.shader_template* **only**. This is an alpha tested shader.

Once you have set up the .shader tag, add the reference to the .bitmap tag set up in the previous step. Then it will look like this:

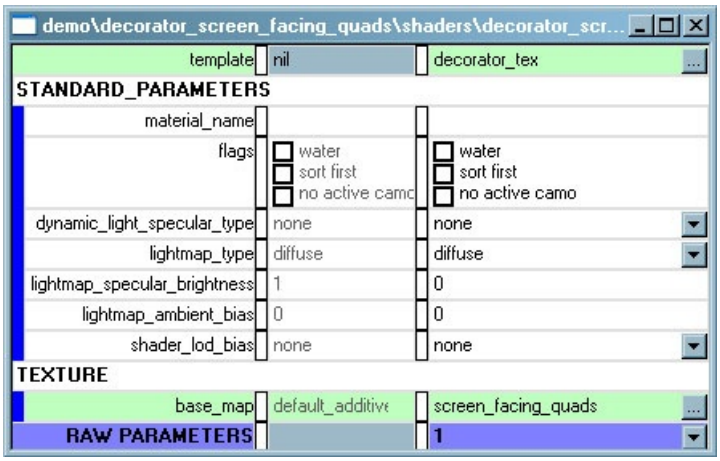


Fig 4. Newly created .shader tag with a reference to the .bitmap tag from the previous step.

> **NOTE**
>
> None of the standard parameters apply to decorators.

SETTING UP THE .DECORATOR_SET TAG

To create the .decorator_set tag: in Guerilla select **File -> New** and set **Group = "decorator_set'**. This will create an empty .decorator_set tag.

SHADERS BLOCK

In the SHADERS block, choose **Add** and then reference the .shader tag.

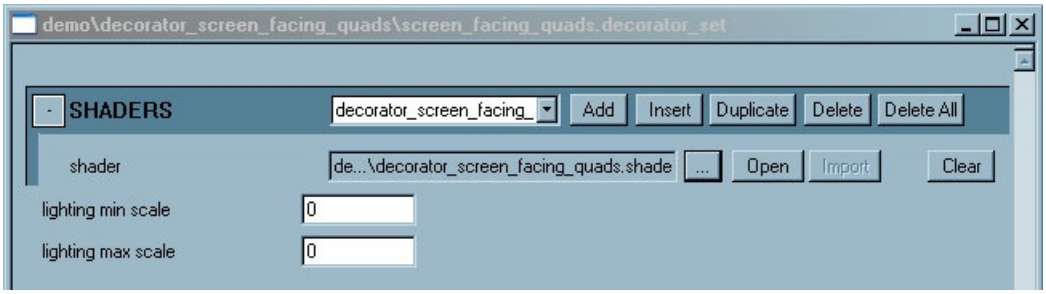


Fig 5. Add a reference to the new .shader tag in the Shaders block of the .bitmap tag.

Next, you'll need to add at least one **decorator class**. Decorator sets can contain multiple classes but for this example we'll make just one class: **screen facing quad**.

CLASSES BLOCK

In the CLASSES block, choose **Add**. The **name** field is what this set of decorator models will be shown as in Sapien.

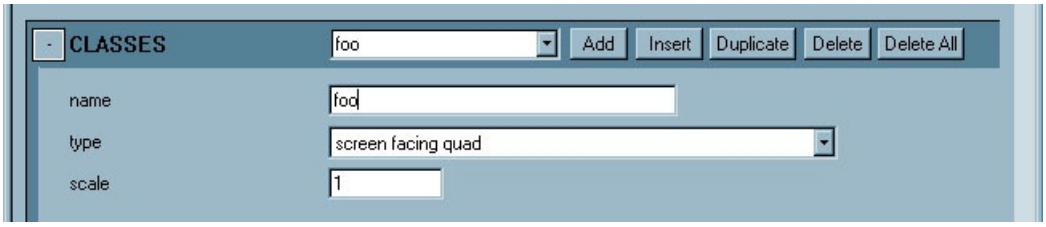


Fig 6. Add a new class and set the type to Screen Facing Quad.

Next we'll add one entry in the PERMUTATIONS block for each sign. Let's start with the **grey plant** image, which is index # 0.

PERMUTATIONS BLOCK

In the PERMUTATIONS block, choose **Add**. The **name** field is what this particular instance will show as in Sapien. Note that the flags "align to normal" and "upright" do not apply to screen facing quads. The **shader** field will allow you to pick any shader defined at the top of the tag in the SHADERS block. For this example, **grey_plant** will fade at a close distance. It is referenced by index #0. It has a distribution weight = 1; this will be compared to the weight of the other permutations when we are putting down the **foo** decorator set in a random application.

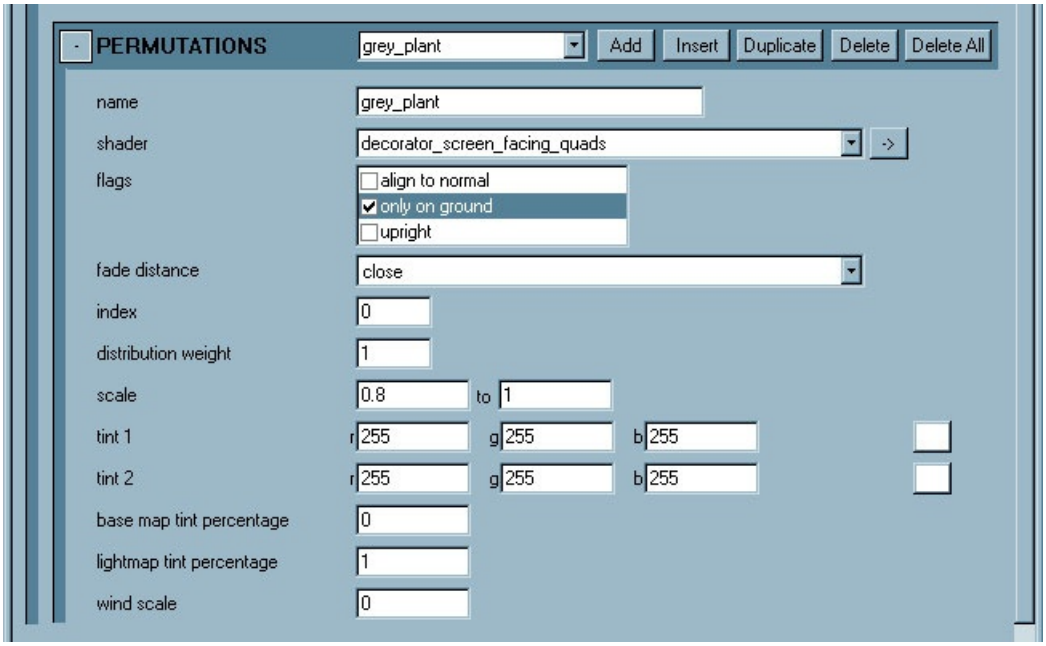


Fig 7. Add the grey_plant to the Permutations block.

Repeat this process for **green plant** (index #1) and **gray rocks** (index #2). Remember that you can use the "Duplicate" button to make this process easier.

READY TO GO!

This decorator set is now ready to be placed in Sapien.



Fig 8. See your decorator set in game.

H2 Decals Overview

11/23/2022 • 2 minutes to read

We have 4 kinds of decals in Halo 2. Each was created to solve different problems.

1. wrapped (Halo 1)
2. decorator quad
3. decorator projected
4. overlay geometry.

Wrapped

- **Description**: Geometry projects over environment and poops directly beneath them and "properly" wrap the adjacent geometry.
- **Implementation**: Placed in Sapien or created from an effect.
- **Render modes**: alpha blends, multiplies, effects: (add, subtract, min, max, color blend)
- **Strengths**: Only decals that wrap. Never fade. Most current render modes.
- **Weaknesses**: More expensive because Sapien ones never fade. Wrap is sometimes not correct looking.
- **Lighting**: We are going to make the alpha blended ones lit by the lightmaps.
- **Used for**: Blast marks, scratches, blood, painted signs big enough you need wrapping or don't want to fade.

Decorator Quad

- **Description**: A planar quad appears with the bitmap registration point at the location picked on the environment or poop in sapien, aligned to the surface's normal. Fade at distance.
- **Implementation**: Placed in Sapien by clicking a point on the structure or poop.
- **Render modes**: alpha blend (we could add more if needed, but we want to keep the number of shaders down).
- **Strengths**: Cheapest of all decals.
- **Weaknesses**: Need to be small enough to fit on a struture/poop plane because they don't conform to geometry and for you to not notice them fade at distance.
- **Lighting**: We are going to make the alpha blended ones lit by the lightmaps.
- **Used for**: Lots of smallish quads you can put everywhere.

Decorator Projected

- **Description**: This geometry conforms to the structure or poop by projecting into a box and creating a bunch of sub-structure-triangles. Fade at distance.
- **Implementation**: Geometry is created by a projector box onto geometry in Sapien.

- **Render modes**: alpha blend (we could add more if needed, but we want to keep the number of shaders down)
- **Strengths**: Great for putting detail on complicated geometry.
- **Weaknesses**: Does not wrap, only projects from the box.
- **Lighting**: We are going to make the alpha blended ones lit by the lightmaps.
- **Used for**: Grime in corners or edges, applying to complicated geometry

Overlay Geometry

- **Description**: Geometry hand crafted, for that beechwood aged flavor.
- **Implementation**: Created in DCC, assigned an overlay shader
- **Render modes**: double multiply (we can easily add more modes that are not alpha blended)
- **Strengths**: Complete control over the geometry created. Never fade.
- **Weaknesses**: Takes a lot of time to create wrapping.
- **Lighting**: These get applied after lighting, lighting never alters it.
- **Used for**: Whatever (non-lightmap-lit) use you want

DYNAMIC LIGHTS AND DECALS

Dynamic lights will not really affect any of these. We made this tradeoff because of the shader LOD system, and do not have time to do another system where dynamic lights can affect them. You will, however, have a choice by render mode whether a decal gets rendered before or after dynamic lights. Pre-dynamic-light ones will be slightly faded out by the light. Post dynamic light ones will look like they are lit by the dynamic light if they are multiplicative, but alpha-blended ones will not be affected at all by the dynamic light.

Crates

11/23/2022 • 3 minutes to read

A **.crate** tag is an **object** tag, like **.biped**, **.scenery** or **.vehicle**.

Crate tags are used for passive objects that require Havok simulation (that is, objects that can be moved in the game - like crates).

The **.crate** tag is created in Guerilla using the **File -> New** command.

.crate TAG

The parameters in the top section of the tag are common to all object tags.

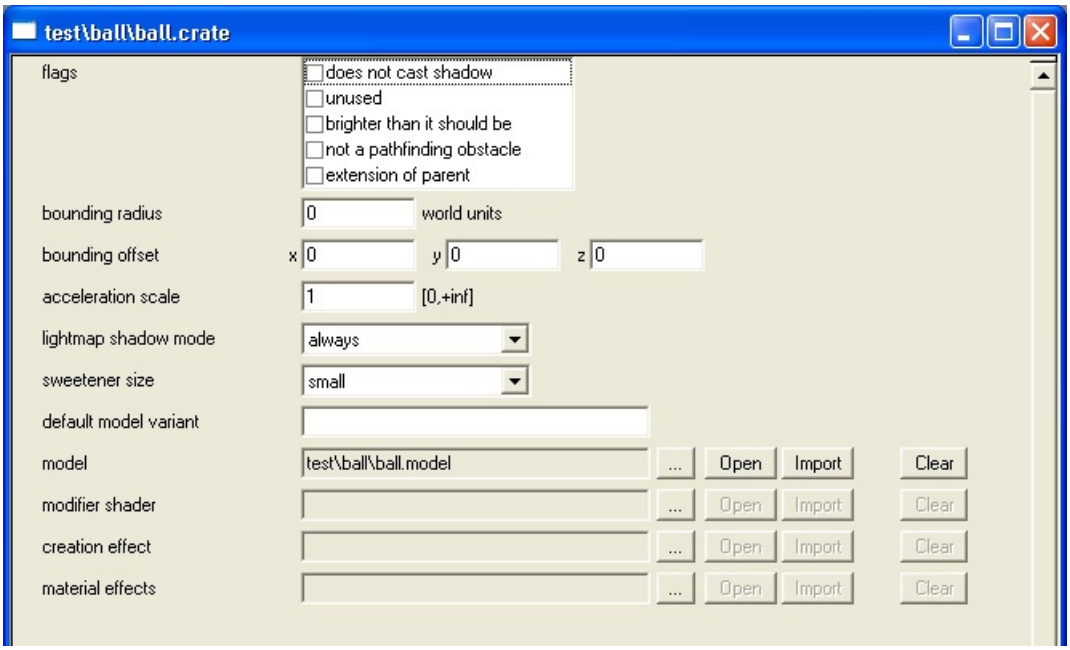


Fig 1. List of common object tag parameters.

- **flags | does not cast shadows**: -
- **flags | unused**: -
- **flags | brighter than it should be**: -
- **flags | not a pathfinding obstacle**: -
- **flags | extension of parent**: -
- **bounding radius**: sphere used to calculate whether the object should be rendered.
- **bounding offset**: offset from origin point of object.
- **acceleration scale**: [marine 1.0, grunt 1.4, elite 0.9, hunter 0.5, etc].
- **lightmap shadow mode**: default | never | always.
- **sweetner size**: -
- **default model variant**: default variant as set in the **.model** tag.

- **model**: link to .model tag.
- **modifier shader**: -
- **creation effect**: -
- **material effects**: -

AI PROPERTIES

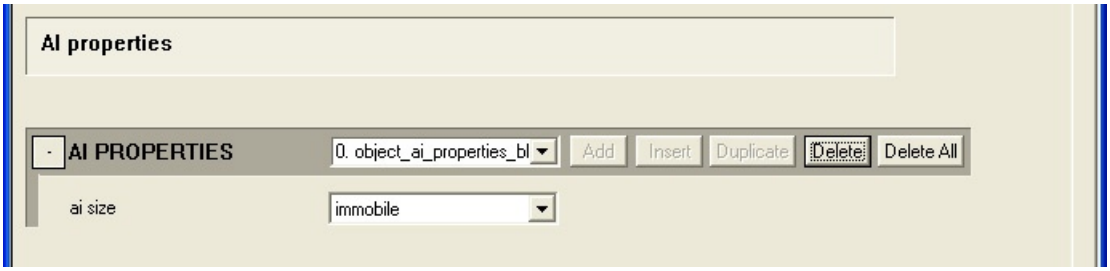


Fig 2. AI Properties tag block.

EXPORT TO FUNCTIONS

Links game data into the object tag.

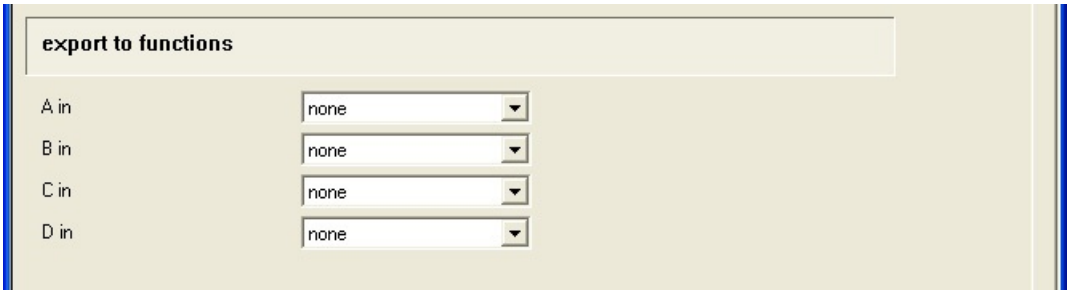


Fig 3. Export to functions section.

A in: pull-down list of available game data such as body vitality, shield vitality and shield stun.

B in: same description as A in.

C in: same description as A in.

D in: same description as A in.

APPLYING COLLISION DAMAGE

Zero will take default settings from **globals.global**. Nothing needs to be set here for the object to function in the game.

This section modifies how the object applies damage to other objects in a collision.



Fig 4. Applying Collision Damage section.

Apply collision damage scale: modifies how the object applies damage to other objects in a collision. Range: ? [O means 1. 1 is standard scale. Some things may want to apply more damage]

GAME COLLISION DAMAGE PARAMETERS

Zero will take default settings from **globals.global**. Nothing needs to be set here for the object to function in the game.

This section determines the threshold and scale for how the object applies and receives damage in a collision with other objects.

Game damage is equal to the magnitude of the velocity of the object being run into divided by the maximum of the magnitude of the velocity of the object or the magnitude of the velocity of the object being run into.

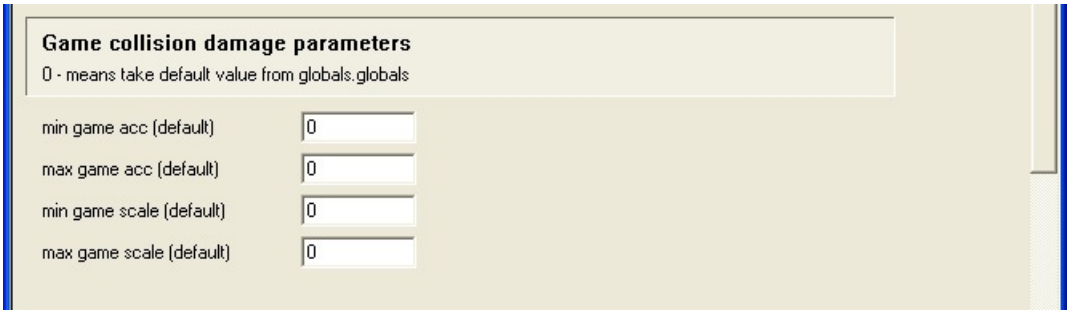


Fig 5. Game Collision Damage Parameters.

min game acc (default): minimum threshold for game damage to occur. Range: zero to infinity.

max game acc (default): maximum threshold for game damage to occur. Range: zero to infinity.

min game scale (default): minimum scale factor for game damage when it does occur. Range: zero to one.

max game scale (default): maximum scale factor for game damage when it does occur. Range: zero to one.

ABSOLUTE COLLISION DAMAGE PARAMETERS

Zero will default to settings in **globals.global**. Nothing needs to be set here for the object to function in the game.

This section determines the threshold and scale for how the object receives damage in a collision with other objects.

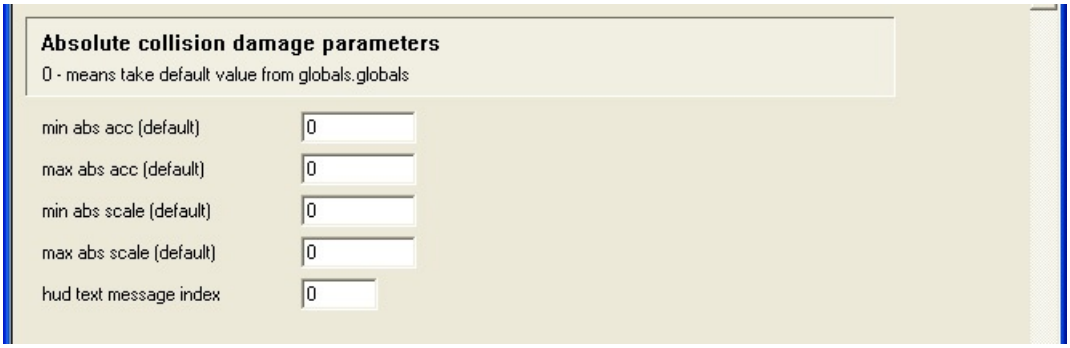


Fig 6. Absolute Collision Damage Parameters.

min abs acc (default): minimum threshold for absolute damage to occur. Range: zero to infinity.

max abs acc (default): maximum threshold for absolute damage to occur. Range: zero to infinity.

min abs scale (default): minimum scale factor for absolute damage when it does occur. Range: zero to one.

max abs scale (default): maximum scale factor for absolute damage when it does occur. Range: zero to one.

hud text message index: N/A.

ATTACHMENTS

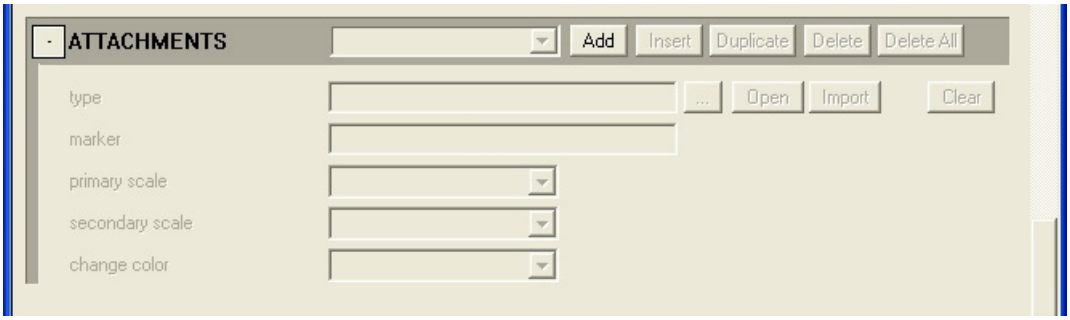


Fig 7. Attachments Section.

FUNCTIONS

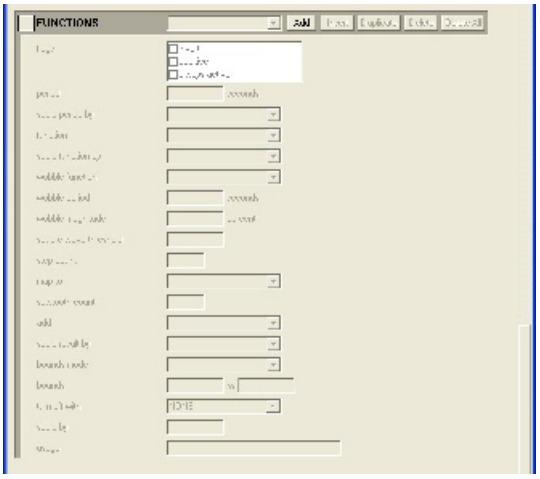


Fig 8. Functions Section.

CHANGE COLORS

Colors generated in the object tag can be used to color the render model. These can be used for random variation, color permutations or team colors.



Fig 9. Change Colors Section.

weight: determines the relative chance of a permutation being chosen.

color lower bound: one end of the color range.

color upper bound: the other end of the color range.

variant name: Permutations can be named in the variant name. That permutation will only show up in a variant of the same name (set up in the .model tag, in the VARIANT: name field - the names must match).

PREDICTED RESOURCES



Fig 10. Predicted Resources Section.

Audio

11/23/2022 • 13 minutes to read

Overview

- Music is used to set the mood, to provide a dramatic component to game mechanics like combat and exploration, and to reward the Player for successes.
- Dialog is used unfold the story, provide characterization and draw the Player into the experience.

DEFINITIONS

The following details all of the various types of sound assets in the game and their function. There are occasional references to implementation for the purpose of describing audio types.

Linear Audio

Any sound used in scripted or pre-rendered scenes that have a set, pre-determined duration is considered linear.

Interruptive Cinematics

Linear sound effects, music and dialog will be mixed to picture in post-production and delivered in the final format, stereo or surround.

Linear Music

Music that is traditionally produced, recorded and mixed to a specific length will usually be used to post score an interruptive cinematic, event, or used as a stand alone piece that might play during a splash screen or for marketing purposes. Think "Soundtrack CD".

Linear Dialog

A traditional script with characters and storyboard that is recorded and then used as an element in any interruptive cinematic. These are scenes that have a set duration that will not change during game play (or after the final mix).

Dynamic Sound Effects

Any sound that responds to a game play condition or event is considered to be Dynamic.

Dynamic sounds can be occluded by geometry, they are spatially located, they are effected by physics (reverb, Doppler shift), they have weighted permutations, they must be assigned a priority and may be effected by special characteristics (i.e. an engine sound will change with the rpm's, or gear shifting, the volume of the sound of a bouncing shell casing will change with the velocity of impact etc.)

Impulse Sounds

Impulse Sounds are triggered sounds that always play all the way through without interruption and usually include many variations.

Footsteps, gunshots, explosions and bullet impacts are examples of Impulse Sounds.

Impulse sounds may have two or more permutations which can be weighted.

Animation Key Frame Sounds

Sounds that are played when an animation hits a key frame.

Impact Sounds

Sounds that are played whenever something impacts a surface. The volume of impact sounds should scale with velocity and distance

Impact Sounds that reflect the material type of the impacted surface are called Sweeteners.

Looping Sounds

Looping Sounds are multi-sectioned sounds that play for an unspecified length of time, usually determined by the Player's actions or by Global Functions.

Looping Sounds are constructed in three sections; the **Intro** which is played only once as the sound starts, the **Loop** which consists of multiple and randomly chosen sections that seamlessly connect to each other and is played for the duration of the sound, and the **Outro** which can either be a smooth transition from the **Loop** or interruptive depending on game conditions and is played when the sound ends.

The **Loop** may have two or more permutations which can be weighted. It may also have several tracks which can be turned on and off independently but are automatically synchronized.

Environmental Sounds

Ambient Sounds are Looping Sounds that are associated with a gameplay area such as a room or outdoor location instead of specific objects, so they are not spatially located.

Ambient Sounds

Ambient Sounds are pre-mixed, pre-spatialized sounds that do not change during game play (i.e. the background sound of an interior room, cavern or forest.)

Ambient Sounds will usually be stereo or surround.

Detail Sounds

Detail Sounds are discreet in nature, like Impulse Sounds, but are considered looped because they are played again after some random time has passed (i.e. bird sounds in the forest, telemetry sounds in a computer).

Detail Sounds are randomly spatialized according to variable parameters.

Attached Sounds

Audio can be attached to specific points in the environment. Audio can also be attached to projectiles, effects, animations, devices, and particles.

Game Mechanics

Health sounds, low time sounds, power-ups, game timers, etc.

Scripted Events

These are what are currently being referred to as Non-Interruptive Run-Time cinematics

Cascading Sounds

Cascading Sounds are a special group of sounds that will be used to simulate events that are impossible to produce without special considerations (i.e. breaking glass, falling gravel, hundreds of mousetraps being sprung by ping pong balls.) Rather than attempting to create a cacophony of sound by multiplying single sound events, we step up from simple to more complex sound files based on the number of game events that are being called. In the example of breaking glass we will step up from the sounds of small_glass_crash to medium_glass_crash to huge_piles_o_glass (in stereo) as more glass is broken simultaneously. This technique will produce the desired overwhelming effect and still give the illusion of synchronization.

Dynamic Music

Dynamic music consists of Looping Sounds of variable length that will be affected by game play.

Dynamic Music consists of an **Intro**, a **Loop** and an **Outro**, but it also has the capability to respond to dynamically changing game conditions.

Atmospheric Music

Similar to ambient sounds in construction but consisting of more musical or non-reality based sound content. Atmospheric music can evoke specific moods or emotions without using what would be considered traditional elements of music. Still dynamic but with less emphasis on interactivity

Thematic Music

This music will be more traditional in nature. Containing recognizable themes and rhythmic elements Thematic Music will immediately alert the player that there is music playing. Unique themes could be associated with characters, locations, and events.

Dynamic Dialog

Pre-recorded lines of dialog that can be triggered during game play.

Scripted Dialog

These are lines designed to play only once during the game. They are tied to a specific game event or scene that may or may not occur depending upon the importance to advancing the story. They can be attached to a specific character or character class i.e. Master Chief talking to Captain Keyes or a Elite talking to a Grunt.

Conditional Dialog

These are lines designed to play numerous times during the game. They can be triggered by several different conditions such as events, locations, or game state. They might be interruptible or even play simultaneously with other character's lines. It's important to record several appropriate permutations that carry the same meaning while keeping the sense of randomness and variation. They can be attached to a specific character or character class i.e. taunts, hurt sounds etc, ally getting killed, killing friends, etc. It's vitally important during casting and recording to keep track of when and where the actors are performing which kind of dialog. Scripted dialog performed by one actor might be followed by generic dialog performed by a different actor but emanating from the same character. Anticipating the number of repetitions of any piece of generic dialog heard during a player's experience of a game is also of vital importance.

Implementation / audio engine tools

What follows is the specific layout of how audio will be implemented into the game. We essentially have three layers: **Soundfiles**, **Soundtags**, and **Looping Sounds**.

Soundfiles

This is the raw data, and is irrelevant to anyone other than the audio team beyond the fact that the next layer refers to them. These files will be 16-bit, 22050 kHz or higher, and can be either mono, stereo, or 5.1.

Soundtags

Any time a Soundtag is called, a single permutation of that Soundtag is played.

Sound Designer will be able to assign many Soundfiles to each sound, and weight each permutations probability of being selected.

Soundfiles assigned to each Soundtag can be either mono or stereo but not both.

The following are the controls we need to have at this layer:

Permutation selection method

Random – Randomly chosen from a weighted list. **This is the Default Choice**

Random, No Repeats – Randomly chosen from a weighted list, but plays all the permutations before repeating

Maximum Number

If enabled, we can set the maximum number of times a Soundtag can be called in a given period of time.

However, if the Soundtag is being used as part of a Looping Sound this value will be ignored.

Pitch

Assignable pitch level at which the Soundtag will play.

Assignable pitch range can be assigned for each individual permutation

Volume

Assignable volume level at which the Soundtag will play

Assignable volume range can be assigned for each individual permutation.

Occlusion Cap

Occlusion can be capped to a level below which occlusion will stop. This will prevent audio from being completely cut off due to excessive layers of occluding material between the player and the audio location.

Priority

Priority can be assigned here in four levels: Game Essential, Dangerous, Relevant, Atmospheric

Cascading Sound Controls

There will be mutiple levels to a Cascading Sound. If the Soundtag (example is glass breaking in a three level cascade) is triggered more than a given number of times in a given period (adjustable) then the next time it is triggered it will trigger the level two Soundfile. If the Soundtag continues to be triggered (now playing level two Soundfile) more than a given number of times in a given period (adjustable) then the next time the Soundtag is triggered it will trigger the level three Soundfile up to a given number of repetitions. This should look something like:

-	-	-
Soundfile one	Number of repetitions	Time period
Soundfile two	Number of repetitions	Time period
Soundfile three	Number of repetitions	

DSP

We will be able to send all of the audio from a Looping Sound to DSP.

Defaults will be no buses assigned and 50% mix amount.

Buses

We can route and re-route audio through as many effects as we want, we should have some way to select what effect(s) we want to be applied.

Mix Amount

We will be able to determine what amount of each effect is mixed in with the original sound.

Auditioning

We need to have the ability to audition the Soundfiles as we import them, and the Soundtag itself playing back with the parameters we have set.

Importing groups of Audio

If when importing audio into a Soundtag we select a folder that contains a number of audio files, all files in the folder will be imported as permutations into the Soundtag.

If we select a folder with a series of nested folders containing varying numbers of audio files, when we import each nested folder will be imported into a separate Soundtag with the files therein as permutations.

Looping Sounds

This is a layer where the designer can utilize various audio elements for use in Dynamic Music and all flavors of Dynamic Sound Effects.

The designer will attach Soundtag in the various areas detailed below, i.e. we will never be assigning Soundfiles directly to the Looping Sounds area. Looping Sounds refer to Soundtags, which in turn refer to the Soundfiles.

The following are the controls we need to have at this layer:

Tracks

A Soundtag will be associated with each track as follows:

Intro

This is a Soundtag that begins the Looping Sound.

Looping Tracks

Ideally we will have several of these to do multi-track mixing "on the fly" in relation to gamestate. The Looping Track will play directly after the Intro is completed. If there is no Intro the Looping Sound would begin with the Looping Track(s)

Outro

This is a Sound that ends the Looping Sound. Ideally we could have multiple Outros selectable by gamestate.

Detail

The Soundtag attached in this area would consist of multiple permutations (such as computer telemetry audio). Detail controls will be:

Time Range

The range of time during which another detail sound will be triggered.

Panning/Positioning

The designer sets the range inside which yaw, pitch, and distance are randomly selected.

Interrupt On Stop

This will be a checkbox that allows the overall Looping Sound to be interrupted wherever it is currently playing (and jump to Outro or just stop if no Outro). If this box is unchecked the Looping Sound will finish playing the current loop and then proceed to the Outro. This should default to unchecked (non-interrupt).

Looping Sound Transition

In relation to gamestate we will be able to move from one piece of music (Looping Sound) to another. For instance a piece of music is playing in the Looping Track area above. As a gamestate parameter changes (player has low health) we could change to a new piece of music in one of two ways:

1. If Interrupt on stop is checked, the music would immediately switch to the Intro of the new Looping Sound.
2. If Interrupt on Stop is not checked, at the end of loop that is currently playing the code would switch to the Intro of the new Looping Sound.

Auditioning

We need to have the ability to audition the Soundtags as we import them, and the Looping Sound itself playing back with the parameters we have set.

Implementation / environment / geometry

Both Soundtags and Looping Sounds can be attached to the following areas:

Geometry

Rooms, hallways, canyons etc will be able to have a Looping Sound attached to them. We also need to be able to subdivide the given Geometry. For example if a tunnel opens up into a large cavern and the combination of the two is considered one piece of geometry we would like to be able to divide the current geometry between the tunnel and cavern so we can attach different audio to those two distinct areas.

Environment

We need to be able to have the ability to place Looping Sounds spatially (with x,y,z coords). On these Soundpoints we need to be able to have alterable shapes (spheres, cones, cylinders, etc.) in which the sound will play at its maximum level, and an additional distance assigned beyond that shape between which the volume will fade to zero.

Particles

When a given particle is present, we will be able to assign a Soundtag to play when the particle impacts (i.e. shrapnel or shell casings).

Projectiles

When a given projectile is present, we will be able to assign a Looping Sound to play while the projectile exists.

Effects

When a given effect is present, we will be able to assign either a Soundtag or Looping Sound to play while the effect is present. Soundtags would be used in cases such as explosions and Looping Sounds would be used for persistant sounds (such as the fire created from an explosion).

Impacts

Audio will be triggered for various impacts based on the material that is moving and the surface material being impacted. This audio will respond to velocity data and can be made up of two distinct Soundtags: a material type Soundtag and a surface type Soundtag (i.e. rifle falling on grass triggers rifle_fall and med_grass_impact).

Animations

Audio can be added to animations via frame number. There will be a way to audition animations and the associated audio files

Vehicles

When a vehicle is at rest (not moving), it will play an idle Looping Sound.

When the vehicle is moving there will be multiple Looping Sounds utilized that respond to conditions such as RPM, acceleration, and deceleration.

Scripted Events

We will able to assign Soundtags or Looping Sounds along with scripted events and pass along volumes, fade times, and other parameters to control audio during scripted events.

Sound Features

Global Fade

An automatic fade of all audio at the end of level or a scripted event, and a partial fade under menu or interface screens.

Sound Debug Mode

In the Sound Debug Mode all the sounds being played are printed on-screen along with attributes (configurable). All audio in use and all attributes can also be dumped to a tab-delimited text file.

Occlusion

Occlusion should not be considered when the audio is passing from non-playable space into playable space. For

example, an audio source placed inside environmental geometry that looks like an air conditioner should not be occluded by the geometry that makes up the air conditioner.

Test Map

A Test Map specially made for testing audio on materials, weapons, characters, environments etc. should be created.

Device Home

11/23/2022 • 2 minutes to read

Device Machines

Motorized, animated, objects placed in the game that either react to the player, or that the player can interact with. Below, you can find detailed information on all of the properties of device machines, as well as step by step setup instructions to get you going quickly.

Device Controls

Device Controls are used to activate or deactivate device machines.

Device Machines

11/23/2022 • 12 minutes to read

Device Machines are motorized, animated, objects placed in the game that either react to the player, or that the player can interact with. Below, you can find detailed information on all of the properties of device machines, as well as step by step setup instructions to get you going quickly.

THE BASICS

- The animation file for your device machine needs to be exported as a **.jmo**. None of the other animation formats will work.
- When creating the animation for your device machine, do not leave any static frames at the beginning or end! This causes problems later when adding sound effects.
- If you want the object's bounding sphere to move with it when it animates in game (and thus retain physics and collision), you need to animate the root node for the object (not any of the lower frame nodes).
- When you export your animation, it needs to be named **device position.jmo**. If the file is not named this way, it will not function! For position animations (which are what most device machine animations will be), our engine animates frame #0, but drops the last frame of the animation. For overlay animations, our engine doesn't play frame #0.

STEP-BY-STEP SETUP

1. Create and import the geometry and animation files for your device machine. You will need render, collision, physics, and animations. Also, make sure you name the animation file "**device position.jmo.**" If you don't, the animation will not play.
 - When creating the animation for your device machine, do not leave groups of static frames at the beginning or end! This causes problems later when adding sound effects.
2. Create any .shader tags for the materials you assigned to your model in the 3D Program of your choice. Re-import.
3. Create a new .model tag. Link all of the other tags for your object to your new .model tag. Make sure you set the material types in your .model tag as well. Save it.
4. In the *Model* tag block of the .model tag, click the Browse (...) button next to the **Animation** text box and link the .model_animation_graph tag for your device machine to your .model tag.
5. Create a new .device_machine tag. Save it to a location of your choosing.
6. Link your .device_machine tag to the .model tag for your object. This field is in the topmost section of your .device_machine tag (labeled *Model*) - simply click on the ... button and browse to the .model tag for your object.
7. Scroll down to the bottom of the .device_machine tag to a block labeled *Device*. There are many different properties to check depending on the type of device machine you're trying to build (see below for detail on each property). To get your device machine up and running, the most important field to enter a number in is **Position Transition Time**. This is the amount of time it will take your device machine to go from closed to open (or off to on). Enter a number (in seconds) here now.
8. Next, find the **Automatic Activation Radius** field. This is the distance (in world units) the player needs to be from the object for it to activate automatically. Think of this in the context of an automatic door - when the player gets close, it opens. Enter a number here now. If you don't want your device to activate automatically, but want to set up a button press or other event which will activate it, check out the Device Controls article. Also, if your device is a "door," and you don't want it to activate automatically, you'll need to check the "Does

not operate automatically" flag in Sapien (See below for more info).

9. Now move down to the *Machines* tag block. For **Type**, select *Gear*. Even if you want to make a door, gear is the most useful for initial debugging of your machine because the *gear* type is always on - so you'll instantly know if your object is working. You can go back and reset the type to *door* at any time once you know the device itself (and particularly its animation) works.
10. Open your scenario in Sapien.
11. Add your new device machine to the Palette, and then place one (by right-clicking on the game window) in the scenario.
12. Select the device you just placed (by clicking on its name in the Hierarchy View) and then select the type for it in the Properties Palette. You can find more information on properties set in Sapien in the Properties Set in Sapien section below.
13. That should be everything you need to set up your basic device machine. Save. If you already had your map running, you will need to reset the map before the device machine will begin to function. See below for more details on individual properties or flags you can set for your device.

DEVICE TAG BLOCK

- **Flags**
 - Position Loops: This flag allows the device to change its position to any value at will, but it will always choose the shortest circular path. When the position goes below zero or above one, it wraps around. For example, if you were at position 0.2 and wanted to to to 0.9, the shortest circular distance is to run backwards (.2, .1, 0, .9). This is different from the Gear machine type in that the Gear type always loops continuously in the direction of the animation you set up (it never goes backwards).
 - Allow Interpolation: Allows the device machine to interpolate animations (they don't do this automatically). Using this flag, you can set a series of positions (without animating them in the 3d Program of your choice) and the game engine will animate the device's movements. However, this will cause the animation to lag behind the device's actual position. For an example, think of the Scarab in Halo 2 (which didn't have animations, but was interpolated by the game engine). You should not use this for for things like doors or elevators.
- **Power Transition Time** - When a power group is set up for the device (with other devices, or with a device control), this controls the amount of time it takes for the device to go from its off state to its on state.
- **Power Acceleration Time** - The amount of time it takes for the speed of the power transition to accelerate from 0 to 1 - or decelerate to 0 - when powering up.
- **Position Transition Time** - The amount of time (in seconds) it takes for the device machine to go from a completely closed (off) to a completely open (on) state.
- **Position Acceleration Time** - The amount of time (in seconds) it takes for the device machine to reach it's top speed (or slow down to zero if it is in 'closing' mode).
- **Depowered Position Transition Time** - The time it takes for the device to enter its depowered state (from a powered state). This only matters when the device is part of a power group (set up in Sapien) with other devices or device controls.
- **Depowered Position Acceleration Time** - The amount of time it takes for the device to accelerate from stopped to top speed on its way into the depowered state. This only matters when the device is part of a power group (set up in Sapien) with other devices or device controls.
- **Lightmap Flags**
 - Don't use in Lightmap: When this flag is checked, the device will not be considered when lightmaps are run. So, for example, if the device is covering an area (or blocking light), the lightmapper will run as if the device did not exist and the area will be lit by lights that might otherwise be blocked.
 - Don't use in Lightprobe: When this flag is checked, the lightprobes will not consider the way a device machine is affecting lighting in the area and, therefore, how the player is lit when standing on/around/near the device.

- **Open (up)** - The effect that is played while the device is opening (or moving up in the case of a lift).
- **Close (down)** - The effect that is played while the device is closing (or moving down in the case of a lift).
- **Opened** - The effect that plays while the device is opened/on.
- **Closed** - The effect that plays while the device is closed/off.
- **Depowered** - The effect that plays when power to the device is shut off.
- **Repowered** - The effect that plays when power to the device is turned on.
- **Delay Time** - The time (in seconds) between when the device is activated and when it actually begins to move.
- **Delay Effect** - The effect that plays during the time between when the device has been activated and when it actually begins to move.
- **Automatic Activation Radius** - The size of the radius (in world units) within which the player needs to stand to activate the device.

MACHINE TAG BLOCK

- **Type** - Select the type of device machine you want to create from the drop down list:
 - Door: The device will behave as a door - opening and closing depending on the settings you choose. Doors are set to open automatically when the player comes within their activation radius unless they are part of a position group with a device control(and you set the "Does Not Operate Automatically" flag in Sapien).
 - Platform: A platform behaves the same as a door device except that it will not activate automatically. For platforms to function, they need to be part of a position group with a device control (a button, for example).
 - Gear: The device will work like a gear - constantly performing it's animations in a single direction - never going backwards and only having the ability (via setup in the tag) to speed up or slow down. In this state, the player doesn't need to turn it on or off, or activate it in any way.
- **Flags**
 - Pathfinding Obstacle: Checking this flag prevents AI from pathing onto the device - they will consider it an obstacle and path around it.
 - ... But not when Open: Checking this flag in conjunction with Pathfinding Obstacle will allow AI to path onto or through the device (or space created by the device) when it is open (on) but not when it is closed (off).
 - Elevator: Allows the device to stop and start at multiple points along the way from completely closed to completely open. For example, the silo elevator in 04b_floodlab (Halo 2). The stopping and starting can be accomplished using scripting (see Tyson or Rob for more info on how to do this). For the elevator to work, you also need to enter an Elevator Node below (in the Machine properties).
- **Door Open Time** - The amount of time (in seconds) a door will remain open before closing (unless it has an automatic activation radius which the player is standing within, or if the door is controlled by a device control - in either of those cases, this property is ignored).
- **Door Occlusion Bounds** - This flag activates and de-activates visibility portals within the distance of the door that you specify (in world units). This is used so that things don't render when the door is closed (or render if we need them to so that they don't "pop" in when the door is opened by the player).
- **Collision Response** - How the device responds when it collides with another object.
 - Pause Until Crushed: The device will continue to its normal open or closed position, crushing anything in its path.
 - Reverse Directions: The device will stop and reverse directions when it collides with an object.
- **Elevator Node** - Used in combination with the Elevator flag (see above) to create device machines which can stop at various points between their fully open or fully closed states. NOTE: Greg says this option appears to be disabled in code (so elevators may not work at the moment).

- **Pathfinding Policy** - Sets the way AI will path on, through, or around the device.
 - Discs: pathfinding for this device will be generated using a disc-type style.
 - Sectors: pathfinding for this device will be generated using sectors.
 - Cut_Out: pathfinding for this device will be generated using cut outs.
 - None: no set pathfinding policy. The engine will decide.

PROPERTIES SET IN SAPIEN

The special properties for Device Machines in Sapien are found in the Properties Palette (after you've placed your device machine in the game window).

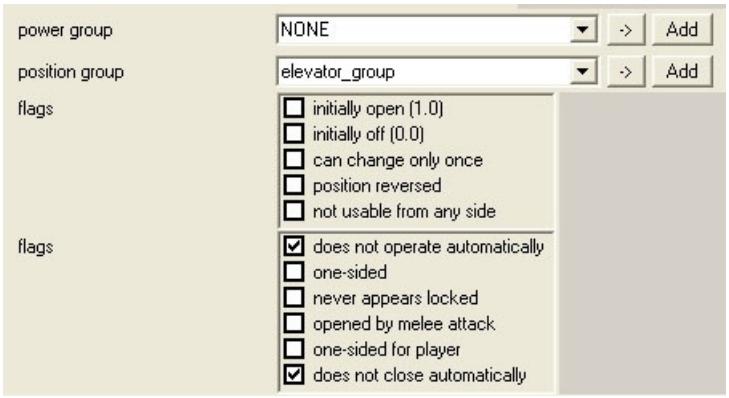


Fig 1 - Device Machine properties set in Sapien

- **Power Group** - Power groups are set up to link device machines and device controls together when the machine has a depowered and powered state.
- **Position Group** - Position groups are links between device controls and device machines. This group controls the animation/movement of the machine. If you want to set up any device machine which is controlled by a button (or any other device control), you must group it in a position group with a device control.
- **Flags**
 - Initially Open (1,0): The device will appear in it's open/on state when the game/level starts.
 - Initially Off (0,0): The device will be in it's depowered/deactivated state when the game/level starts.
 - Can Change Only Once: allows the device machine to change position (open or close, for example) only one time. Once it has moved, it cannot be moved again.
 - Position Reversed: Reverses the position values for the device. So, a door that would normally start out in its open state would start out in its closed position when this flag is checked.
 - Not usable from any side: prevents the player from being able to use the device.
- **Flags**
 - Does Not Operate Automatically: the device will not operate automatically. If this flag is set, the "automatic activation radius" property in Guerilla will be ignored, and you will need to set up a position group with a device control in order to operate the machine.
 - One-Sided: the device will only operate from one side. The side can be set as the positive x-axis of a marker named "one_way" or the x-axis of the door (or other device) itself if no marker exists.
 - Never Appears Locked: For door objects. The device will never report itself as locked to object function callers. In other words, it will never appear locked to the player (appearance-wise).
 - Opened By Melee Attack: when this flag is set, the player will need to melee the device machine for it to enter its opened state.
 - One-sided For Player: The device will only be accesible to the player from one side. AI can access it from either side. See "One-Sided" flag description for more information.
 - Does Not Close Automatically: prevents a device machine from closing automatically. You will need to

set up a position group with a device control (such as a button) to get the device to close.

Device Controls

11/23/2022 • 9 minutes to read

Device Controls are used to activate or deactivate device machines.

THE BASICS

- Device Controls need to be linked to Device Machines using a **Position Group** in Sapien. Position groups are set up so that the position animation will play. Controls and machines can also be linked in a **Power Group** in order to power on or off device machines (power groups will not play position animations).
- Device Controls can not have physics (although they can have collision), so they need to either be placed on a piece of geometry, or linked to an object to give the appearance of having physics.
- Device Controls can not have position animations. If you want to animate your device control, you need to attach it to an object using the objects_attach script command.

STEP-BY-STEP SETUP INSTRUCTIONS

1. Create and import the geometry (render and collision) for your Device Control.
2. Create a new .model tag. Link your render and collision tags to it.
3. At some point, don't forget to create .shader tags for the materials you set in Max and then re-import your model.
4. Create a new **.device_control** tag. Link it to the .model tag you created previously. This field is in the topmost section of your .device_control tag (labeled *Model*) - simply click on the ... button and browse to the .model tag for your object.
5. Scroll down to the very bottom of your device_control tag to the block labeled **Control**.
6. In the *Type* drop down list, select **Toggle Switch**. You can always come back and change this later if you're not intending to build a toggled device.
7. In the *Triggers* When drop down list, select **Touched By Player**.
8. In the *Action String* text box, you can enter any text you'd like to appear when the player gets near the object. This is the text that appears on-screen and tells the player what to do to activate the device. See below for detailed information on how this string works (and how to set it up with a button icon).
9. Now is a good time to save all the tags you just created. Please do so.
10. Open a scenario in Sapien (the one where you'd like to place this control - and a device machine if one is not already present).
11. Add your new device control to the Palette, and then place one (by right-clicking on the game window) in the scenario.
12. Select the device you just placed (by clicking on its name in the Hierarchy View) and then select the type for it in the Properties Palette.
13. In order for your device control to work with a device machine, you need to link them together in a **Position Group**. Click on the *Device Groups* folder in the Hierarchy View and choose **New Instance**. Choose a fitting name for the group.

14. Now go back to the *Controls* folder in the Hierarchy View. Select your Device Control in the Pane on the right. In the *Properties Palette*, assign your control to the new group you just created (Select the name of the group from the **Position Group** drop-down list).

15. If you haven't placed/created a device machine yet, do so now. See the Device Machines article for more detail on how to create them.

16. In Sapien, make sure your device machine has the **Does Not Operate Automatically** flag checked (otherwise it will ignore your new device control).

17. Assign your Device Machine to the same **Position Group** as the one your device control is assigned to.

18. Save and Reset the map. You should now be able to activate/de-activate your device with the device control.

DEVICE TAG BLOCK

- **Flags**
 - **Position Loops**: This is an obsolete property for looping animations. Device controls cannot have position animations.
 - **Allow Interpolation**: This is an obsolete property for interpolating animations. Device controls cannot have position animations.
- **Power Transition Time** - The amount of time it takes for the device control to go from its depowered state to its powered state. The time doesn't affect the behavior of the device machine that the control is linked to. However, device controls can have effects attached to their various states, so this number will affect the amount of time that the effect attached to this state will play. Powered and Depowered states can be set up using a Power Group in Sapien.
- **Power Acceleration Time** - The amount of time it takes for the speed of the power transition to accelerate from 0 to 1 (or decelerate to 0) when powering up. This time doesn't affect the behavior of the device machine that the control is linked to, and device controls cannot have animations of their own. However, device controls can have effects attached to their various states, so this number will affect the amount of time that the effect attached to this state will play. Powered and Depowered states can be set up using a Power Group in Sapien.
- **Position Transition Time** - Device Controls cannot have position animations. However, they can have effects attached to their various states, so this number will affect the amount of time that the effect attached to this state will play. This number should not have any effect on the position transition time of the device machine the control is grouped with.
- **Position Acceleration Time** - Device Controls cannot have position animations. However, they can have effects attached to their various states, so this number will affect the amount of time that the effect attached to this state will play. This number should not have any effect on the position transition time of the device machine the control is grouped with.
- **Depowered Position Transition Time** - The amount of time it takes for the device control to go from its powered state to its depowered state. The time doesn't affect the behavior of the device machine that the control is linked to. However, device controls can have effects attached to their various states, so this number will affect the amount of time that the effect attached to this state will play. Powered and Depowered states can be set up using a Power Group in Sapien.
- **Depowered Position Acceleration Time** - The amount of time it takes for the speed of the power transition to decelerate from 1 to 0 when powering down. This time doesn't affect the behavior of the device machine that the control is linked to, and device controls cannot have animations of their own.

However, device controls can have effects attached to their various states, so this number will affect the amount of time that the effect attached to this state will play. Powered and Depowered states can be set up using a Power Group in Sapien.

- **Lightmap Flags**
 - **Don't use in lightmap**: When this flag is checked, the control will not be considered when lightmaps are run. So, for example, if the control is covering an area (or blocking light), the lightmapper will run as if the control did not exist and the area will be lit by lights that might otherwise be blocked. Or, in another case, a device control might have shaders which use self-illumination or have emissive properties. When this flag is checked, those would be ignored by the lightmapper.
 - **Don't use in lightprobe**: When this flag is checked, the lightprobes will not consider the way a device control is affecting lighting in the area and, therefore, how the player is lit when standing on/around/near the device.
- **Open (up)** - the effect that plays while the control is entering its opened state.
- **Close (down)** - the effect that plays while the control is entering its closed state
- **Opened** - the effect that plays while the control is in its opened/on/activated state
- **Closed** - the effect that plays while the control is in its closed/off/deactivated state
- **Depowered** - the effect that plays while the control is in its depowered state.
- **Repowered** - the effect that plays while the control is in its powered state.
- **Delay Time** - the amount of time that passes between when the player attempts to activate the control and when it actually activates.
- **Delay Effect** - the effect that plays during the control's delay time.
- **Automatic Activation Radius** - size in world units of the space where, when the player steps within it, the control will be automatically activated.

CONTROL TAG BLOCK

- **Type**
 - **Toggle Switch**: if the current position of the device (grouped with the control) is greater than 0.5, then the device is told to go to zero (off or closed). Otherwise, it is told to go to 1 (on or open). Switches the device to the opposite of its current state.
 - **On Button**: this sets the control so that whenever it is pressed, it tells the device to go to 1.0 (fully on).
 - **Off Button**: this sets the control so that whenever it is pressed, it tells the device to go to 0.0 (fully off).
 - **Call Button**: whenever hit, this tells the target machine to go to the position listed in the control definition (this is the Call Value field - enter any number between 0 and 1).
- **Triggers When**
 - **Touched by Player**: the control will activate when the player presses and holds the "X" button when standing within the activation radius (getting the activation message).
 - **Destroyed**: the control will only activate when it has been destroyed.

- **Call Value** - any value between 0 and 1 (0 being off/closed, 1 being on/open). This value is used in conjunction with setting the control type as a Call Button.
- **Action String** - The string that will appear on-screen in the player's HUD telling them how to activate the device. You can type any string you want here, but if you want button icons (or other pictures) to appear, you have to place a special string in the HUD_messages (\main\data\UI\hud_messages) text file and then call the string here.
- **On** - the effect that plays while the control is in its on state.
- **Off** - the effect that plays while the control is in its off state.
- **deny** - the effect that plays when the player is denied activation of the control.

SET DEVICE CONTROL PROPERTIES IN SAPIEN

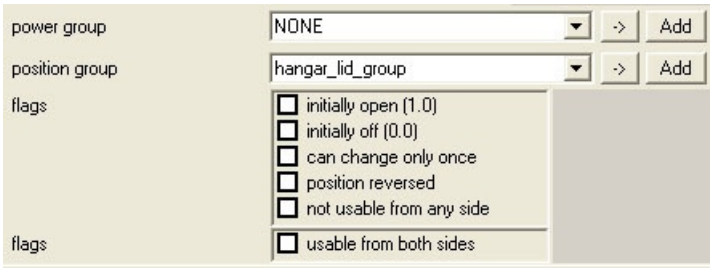


Fig 1 - Sapien Properties for Device Controls

- **Power Group** - Power groups are set up to link device machines and device controls together when the machine has a depowered and powered state.
- **Position Group** - Position groups are links between device controls and device machines. If you want to use a device control (such as a button) to control a device machine, you need to make a new position group and add both the device control and the device machine to it.
- **Flags**
 - **initially open**: The control will appear in it's open/on state when the game/level starts.
 - **initially off**: The control will be in it's depowered/deactivated state when the game/level starts.
 - **can change only once**: Allows the control to be used only one time (thus preventing the device machine from being used more than once as well).
 - **position reversed**: Reverses the position values for the control. So, a button that would normally start out in its on state would start out in its off state when this flag is checked.
 - **not usable from any side**: This flag sets the control as unusable by the player (although the AI could still use it).
- **Flags**
 - **usable from both sides**: The player will be able to access the control from any side.

H2 HaloScript Reference

11/23/2022 • 41 minutes to read

NOTE

This list may be outdated. You can get the latest list using 'script_doc' in the game console or running the script-doc command in tool.exe to ensure you have a full list.

(begin <expression(s)>)

returns the last expression in a sequence after evaluating the sequence in order.

(begin_random <expression(s)>)

evaluates the sequence of expressions in random order and returns the last value evaluated.

(if <boolean> <then> [<else>])

returns one of two values based on the value of a condition.

(cond (<boolean1> <result1>) [(<boolean2> <result2>) [...]])

returns the value associated with the first true condition.

(set <variable name> <expression>)

set the value of a global variable.

(and <boolean(s)>)

returns true if all specified expressions are true.

(or <boolean(s)>)

returns true if any specified expressions are true.

(+ <number(s)>)

returns the sum of all specified expressions.

(- <number> <number>)

returns the difference of two expressions.

(* <number(s)>)

returns the product of all specified expressions.

(/ <number> <number>)

returns the quotient of two expressions.

(min <number(s)>)

returns the minimum of all specified expressions.

(max <number(s)>)

returns the maximum of all specified expressions.

(= <expression> <expression>)

returns true if two expressions are equal

(!= <expression> <expression>)

returns true if two expressions are not equal

(> <number> <number>)

returns true if the first number is larger than the second.

(< <number> <number>)

returns true if the first number is smaller than the second.

(>= <number> <number>)

returns true if the first number is larger than or equal to the second.

(<= <number> <number>)

returns true if the first number is smaller than or equal to the second.

(sleep <short> [<script>])

pauses execution of this script (or, optionally, another script) for the specified number of ticks.

(sleep_forever [<script>])

pauses execution of this script (or, optionally, another script) forever.

(sleep_until <boolean> [<short>])

pauses execution of this script until the specified condition is true, checking once per second unless a different number of ticks is specified.

(wake <script name>)

wakes a sleeping script in the next update.

(inspect <expression>)

prints the value of an expression to the screen for debugging purposes.

(unit <object>)

converts an object to a unit.

(not <boolean>)

returns the opposite of the expression.

(pin <real> <real> <real>)

returns the first value pinned between the second two

(print <string>)

prints a string to the console.

(players)

returns a list of the players

(volume_teleport_players_not_inside <trigger_volume> <cutscene_flag>)

moves all players outside a specified trigger volume to a specified flag.

(volume_test_object <trigger_volume> <object>)

returns true if the specified object is within the specified volume.

(volume_test_objects <trigger_volume> <object_list>)

returns true if any of the specified objects are within the specified volume.

(volume_test_objects_all <trigger_volume> <object_list>)

returns true if any of the specified objects are within the specified volume.

(list_get <object_list> <short>)

returns an item in an object list.

(list_count <object_list>)

returns the number of objects in a list

(effect_new <effect> <cutscene_flag>)

starts the specified effect at the specified flag.

(effect_new_on_object_marker <effect> <object> <string>)

starts the specified effect on the specified object at the specified marker.

(damage_new <damage> <cutscene_flag>)

causes the specified damage at the specified flag.

(damage_object <damage> <object>)

causes the specified damage at the specified object.

(damage_objects <damage> <object_list>)

causes the specified damage at the specified object list.

(damage_players <damage>)

damages all players with the given damage effect

(object_create <object_name>)

creates an object from the scenario.

(object_create_clone <object_name>)

creates an object, potentially resulting in multiple objects if it already exists.

(object_create_anew <object_name>)

creates an object, destroying it first if it already exists.

(object_create_containing <string>)

creates all objects from the scenario whose names contain the given substring.

(object_create_clone_containing <string>)

creates clones for all objects from the scenario whose names contain the given substring.

(object_create_anew_containing <string>)

creates anew all objects from the scenario whose names contain the given substring.

(object_destroy <object>)

destroys an object.

(object_destroy_containing <string>)

destroys all objects from the scenario whose names contain the given substring.

(object_destroy_all)

destroys all non player objects.

(object_destroy_type_mask <long>)

destroys all objects matching the type mask

(objects_delete_by_definition <object_definition>)

deletes all objects of type <definition>

(object_hide <object> <boolean>)

hides or shows the object passed in

(object_set_shadowless <object> <boolean>)

set/reset shadow castingness of object

(object_set_ranged_attack_inhibited <object> <boolean>)

FALSE prevents object from using ranged attack

(object_set_melee_attack_inhibited <object> <boolean>)

FALSE prevents object from using melee attack

(objects_dump_memory)

debugs object memory usage

(object_get_health <object>)

returns the health [0,1] of the object, returns -1 if the object does not exist

(object_get_shield <object>)

returns the shield [0,1] of the object, returns -1 if the object does not exist

(object_set_shield_effect <object> <real> <real>)

sets the shield response effect (not current shield amount) to a given value over the given number of seconds (cinematic use only, remember to call (object_set_shield_effect 0 0) after use!)

(object_set_physics <object> <boolean>)

prevents an object from running physics or colliding with any other objects

(object_get_parent <object>)

returns the parent of the given object

(objects_attach <object> <string> <object> <string>)

attaches the second object to the first both strings can be empty

(objects_detach <object> <object>)

detaches from the given parent object the given child object

(object_set_scale <object> <real> <short>)

sets the scale for a given object and interpolates over the given number of frames to achieve that scale

(object_set_velocity <object> <real>)

Sets the (object-relative) forward velocity of the given object

(object_set_velocity <object> <real> <real> <real>)

Sets the (object-relative) velocity of the given object

(garbage_collect_now)

causes all garbage objects except those visible to a player to be collected immediately

(garbage_collect_unsafe)

forces all garbage objects to be collected immediately, even those visible to a player (dangerous!)

(garbage_collect_multiplayer)

runs multiplayer garbage collection

(object_cannot_take_damage <object_list>)

prevents an object from taking damage

(object_can_take_damage <object_list>)

allows an object to take damage again

(object_cinematic_lod <object> <boolean>)

makes an object use the highest lod for the remainder of the levels' cutscenes.

(object_cinematic_collision <object> <boolean>)

makes an object not collide with other cinematic collision objects.

(object_uses_cinematic_lighting <object> <boolean>)

makes an object use the cinematic directional and ambient lights instead of sampling the lightmap.

(cinematic_lighting_set_primary_light <real> <real> <real> <real> <real>)

sets the pitch, yaw, and color (red, green, blue) of the cinematic shadowing diffuse and specular directional light.

(cinematic_lighting_set_secondary_light <real> <real> <real> <real> <real>)

sets the pitch, yaw, and color (red, green, blue) of the cinematic non-shadowing diffuse directional light.

(cinematic_lighting_set_ambient_light <real> <real> <real>)

sets the color (red, green, blue) of the cinematic ambient light.

(objects_predict <object_list>)

loads textures/geometry/sounds necessary to present objects that are about to come on-screen

(objects_predict_high <object_list>)

loads textures/geometry/sounds necessary to present objects that are about to come on-screen

(objects_predict_low <object_list>)

loads textures/geometry/sounds necessary to present objects that are about to come on-screen

(object_type_predict_high <object_definition>)

loads textures necessary to draw an object that's about to come on-screen.

(object_type_predict_low <object_definition>)

loads textures necessary to draw an object that's about to come on-screen.

(object_type_predict <object_definition>)

loads textures necessary to draw an object that's about to come on-screen.

(object_teleport <object> <cutscene_flag>)

moves the specified object to the specified flag.

(object_set_facing <object> <cutscene_flag>)

turns the specified object in the direction of the specified flag.

(object_set_shield <object> <real>)

sets the shield vitality of the specified object (between 0 and 1).

(object_set_permutation <object> <string> <string>)

sets the desired region (use "" for all regions) to the permutation with the given name, e.g. (object_set_permutation flood "right arm" ~damaged)

(object_set_region_state <object> <string> <model_state>)

sets the desired region (use "" for all regions) to the model state with the given name, e.g. (object_set_region_state marine head destroyed)

(objects_can_see_object <object_list> <object> <real>)

returns true if any of the specified units are looking within the specified number of degrees of the object.

(objects_can_see_flag <object_list> <cutscene_flag> <real>)

returns true if any of the specified units are looking within the specified number of degrees of the flag.

(objects_distance_to_object <object_list> <object>)

returns minimum distance from any of the specified objects to the specified destination object. (returns -1 if there are no objects to check)

(objects_distance_to_flag <object_list> <cutscene_flag>)

returns minimum distance from any of the specified objects to the specified flag. (returns -1 if there are no objects, or no flag, to check)

(position_predict <real> <real> <real>)

in: x, y, z position. loads textures/geometry/sounds necessary to present locations that are about to come on-screen.

(shader_predict <string>)

in: shader name. loads textures necessary for a shader.

(bitmap_predict <string>)

in: bitmap name. loads all the bitmaps in that bitmap group

(script_recompile)

recompiles scripts.

(script_doc)

saves a file called hs_doc.txt with parameters for all script commands.

(help <string>)

prints a description of the named function.

(random_range <short> <short>)

returns a random value in the range [lower bound, upper bound)

(real_random_range <real> <real>)

returns a random value in the range [lower bound, upper bound)

(physics_constants_reset)

resets all physics constants to earthly values

(physics_set_gravity <real>)

set global gravity acceleration relative to earth

(havok_debug_start)

start up the havok visual debugger

(havok_dump_world <string> <boolean>)

dump the state of the havok world, with our without a movie

(havok_dump_world_close_movie)

end the capture of a havok dump movie

(breakable_surfaces_enable <boolean>)

enables or disables breakability of all breakable surfaces on level

(breakable_surfaces_reset)

restores all breakable surfaces

(recording_play <unit> <cutscene_recording>)

make the specified unit run the specified cutscene recording.

(recording_play_and_delete <unit> <cutscene_recording>)

make the specified unit run the specified cutscene recording, deletes the unit when the animation finishes.

(recording_play_and_hover <vehicle> <cutscene_recording>)

make the specified vehicle run the specified cutscene recording, hovers the vehicle when the animation finishes.

(recording_kill <unit>)

kill the specified unit's cutscene recording.

(recording_time <unit>)

return the time remaining in the specified unit's cutscene recording.

(render_lights <boolean>)

enables/disables dynamic lights

(render_debug_structure_cluster <long>)

enables cluster debugging

(render_debug_structure_fog_plane <long>)

enables fog plane debugging

(render_debug_structure_fog_zone <long>)

enabled fog zone debugging

(render_debug_structure_fog_zone_floodfill <long>)

enables fog zone debugging

(render_debug_structure_opacity <real>)

sets the opacity (0 is default)

(render_debug_structure_non_occluded_fog_planes <boolean>)

controls non-occluded fog plane debugging

(scenery_get_animation_time <scenery>)

returns the number of ticks remaining in a custom animation (or zero, if the animation is over).

(scenery_animation_start <scenery> <animation_graph> <string>)

starts a custom animation playing on a piece of scenery

(scenery_animation_start_at_frame <scenery> <animation_graph> <string> <short>)

starts a custom animation playing on a piece of scenery at a specific frame

(scenery_animation_start_relative <scenery> <animation_graph> <string> <object>)

starts a custom animation playing on a piece of scenery relative to a parent object

(scenery_animation_idle <scenery>)

starts the idle animation (if any) for a piece of scenery

(render_effects <boolean>)

(debug_pvs <boolean>)

displays the current pvs.

(unit_can_blink <unit> <boolean>)

allows a unit to blink or not (units never blink when they are dead)

(unit_open <unit>)

opens the hatches on the given unit

(unit_close <unit>)

closes the hatches on a given unit

(unit_kill <unit>)

kills a given unit, no saving throw

(unit_kill_silent <unit>)

kills a given unit silently (doesn't make them play their normal death animation or sound)

(unit_get_custom_animation_time <unit>)

returns the number of ticks remaining in a unit's custom animation (or zero, if the animation is over).

(unit_stop_custom_animation <unit>)

stops the custom animation running on the given unit.

(custom_animation <unit> <animation_graph> <string> <boolean>)

starts a custom animation playing on a unit (interpolates into animation if last parameter is TRUE)

(custom_animation_relative <unit> <animation_graph> <string> <boolean> <object>)

starts a custom animation relative to some other object (interpolates into animation if last parameter is TRUE)

(custom_animation_list <object_list> <animation_graph> <string> <boolean>)

starts a custom animation playing on a unit list (interpolates into animation if last parameter is TRUE)

(unit_set_actively_controlled <unit> <boolean>)

sets unit's actively controlled flag

(unit_custom_animation_at_frame <unit> <animation_graph> <string> <boolean> <short>)

starts a custom animation playing on a unit at a specific frame index(interpolates into animation if next to last parameter is TRUE)

(unit_is_playing_custom_animation <unit>)

returns TRUE if the given unit is still playing a custom animation

(unit_aim_without_turning <unit> <boolean>)

allows a unit to aim in place without turning

(unit_set_enterable_by_player <unit> <boolean>)

can be used to prevent the player from entering a vehicle

(unit_enter_vehicle <unit> <vehicle> <string>)

puts the specified unit in the specified vehicle (in the named seat)

(unit_board_vehicle <unit> <string>)

Causes the given unit to attempt to board the named seat

(unit_set_emotion <unit> <short>)

sets a unit's facial expression (-1 is none, other values depend on unit)

(unit_set_emotion_animation <unit> <string>)

sets the emotion animation to be used for the given unit

(unit_set_emotional_state <unit> <string> <real> <short>)

sets a unit's facial expression by name with weight and transition time

(unit_enable_eye_tracking <unit> <boolean>)

enable/disable eye aiming on a unit

(unit_in_vehicle <unit>)

returns true if the given unit is seated on a parent unit

(vehicle_test_seat_list <vehicle> <string> <object_list>)

tests whether the named seat has an object in the object list (use "" to test all seats for any unit in the list)

(vehicle_test_seat <vehicle> <string> <unit>)

tests whether the named seat has a specified unit in it (use "" to test all seats for this unit)

(unit_set_prefer_tight_camera_track <unit> <boolean>)

sets the unit to prefer a tight camera track

(unit_exit_vehicle <unit>)

makes a unit exit its vehicle

(unit_exit_vehicle <unit> <short>)

makes a unit exit its vehicle (0 = normal exit to airborne, 1 = ejection, 2 = ejection + death, 3 = exit to ground)

(unit_set_maximum_vitality <unit> <real> <real>)

sets a unit's maximum body and shield vitality

(units_set_maximum_vitality <object_list> <real> <real>)

sets a group of units' maximum body and shield vitality

(unit_set_current_vitality <unit> <real> <real>)

sets a unit's current body and shield vitality

(units_set_current_vitality <object_list> <real> <real>)

sets a group of units' current body and shield vitality

(vehicle_load_magic <unit> <string> <object_list>)

makes a list of units (named or by encounter) magically get into a vehicle, in the substring-specified seats (e.g. CD-passenger... empty string matches all seats)

(vehicle_unload <unit> <string>)

makes units get out of a vehicle from the substring-specified seats (e.g. CD-passenger... empty string matches all seats)

(unit_set_animation_mode <unit> <string>)

this unit will assume the named animation mode

(magic_melee_attack)

causes player's unit to start a melee attack

(vehicle_riders <unit>)

returns a list of all riders in a vehicle

(vehicle_driver <unit>)

returns the driver of a vehicle

(vehicle_gunner <unit>)

returns the gunner of a vehicle

(unit_get_health <unit>)

returns the health [0,1] of the unit, returns -1 if the unit does not exist

(unit_get_shield <unit>)

returns the shield [0,1] of the unit, returns -1 if the unit does not exist

(unit_get_total_grenade_count <unit>)

returns the total number of grenades for the given unit, 0 if it does not exist

(unit_has_weapon <unit> <object_definition>)

returns TRUE if the <unit> has <object> as a weapon, FALSE otherwise

(unit_has_weapon_readied <unit> <object_definition>)

returns TRUE if the <unit> has <object> as the primary weapon, FALSE otherwise

(unit_lower_weapon <unit> <short>)

lower the units weapon over x ticks

(unit_raise_weapon <unit> <short>)

raises the units weapon over x ticks

(unit_animation_forced_seat <string>)

all units controlled by the player will assume the given seat name (valid values are 'asleep', 'alert', 'stand', 'crouch' and 'flee')

(unit_doesnt_drop_items <object_list>)

prevents any of the given units from dropping weapons or grenades when they die

(unit_impervious <object_list> <boolean>)

prevents any of the given units from being knocked around or playing ping animations

(unit_suspended <unit> <boolean>)

stops gravity from working on the given unit

(units_set_desired_flashlight_state <object_list> <boolean>)

sets the units' desired flashlight state

(unit_set_desired_flashlight_state <unit> <boolean>)

sets the unit's desired flashlight state

(unit_get_current_flashlight_state <unit>)

gets the unit's current flashlight state

(unit_add_equipment <unit> <starting_profile> <boolean>)

adds/resets the unit's health, shield, and inventory (weapons and grenades) to the named profile. resets if third parameter is true, adds if false.

(device_set_never_appears_locked <device> <boolean>)

changes a machine's never_appears_locked flag, but only if paul is a bastard

(device_set_power <device> <real>)

immediately sets the power of a named device to the given value

(device_get_power <device>)

gets the current power of a named device

(device_set_position <device> <real>)

set the desired position of the given device (used for devices without explicit device groups)

(device_get_position <device>)

gets the current position of the given device (used for devices without explicit device groups)

(device_set_position_immediate <device> <real>)

instantaneously changes the position of the given device (used for devices without explicit device groups

(device_group_get <device_group>)

returns the desired value of the specified device group.

(device_group_set <device_group> <real>)

changes the desired value of the specified device group.

(device_group_set_immediate <device_group> <real>)

instantaneously changes the value of the specified device group.

(device_one_sided_set <device> <boolean>)

TRUE makes the given device one-sided (only able to be opened from one direction), FALSE makes it two-sided

(device_operates_automatically_set <device> <boolean>)

TRUE makes the given device open automatically when any biped is nearby, FALSE makes it not

(device_group_change_only_once_more_set <device_group> <boolean>)

TRUE allows a device to change states only once

(cheat_all_powerups)

drops all powerups near player

(cheat_all_weapons)

drops all weapons near player

(cheat_all_vehicles)

drops all vehicles on player

(cheat_teleport_to_camera)

teleports player to camera location

(cheat_active_camouflage)

gives the player active camouflage

(cheat_active_camouflage_by_player <short>)

gives a specific player active camouflage

(cheats_load)

reloads the cheats.txt file

(ai_enable <boolean>)

turns all AI on or off.

(ai_enabled)

returns whether AI is turned on or off.

(ai_dialogue_triggers <boolean>)

turns impromptu dialogue on or off.

(ai_grenades <boolean>)

turns grenade inventory on or off.

(ai_get_object <ai>)

returns the unit/object corresponding to the given actor

(ai_get_unit <ai>)

returns the unit/object corresponding to the given actor

(ai_attach <unit> <ai>)

attaches the specified unit to the specified encounter.

(ai_attach_units <object_list> <ai>)

attaches the specified list of units to the specified encounter.

(ai_detach <unit>)

detaches the specified unit from all AI.

(ai_detach_units <object_list>)

detaches the specified list of units from all AI.

(ai_place <ai>)

places the specified squad on the map.

(ai_place <ai> <short>)

places the given number of members of the specified squad.

(ai_place_in_vehicle <ai> <ai>)

places the specified squad (1st arg) on the map in the vehicles belonging to the specified vehicle squad (2nd arg).

(ai_kill <ai>)

instantly kills the specified encounter and/or squad.

(ai_kill_silent <ai>)

instantly and silently (no animation or sound played) kills the specified encounter and/or squad.

(ai_erase <ai>)

erases the specified encounter and/or squad.

(ai_erase_all)

erases all AI.

(ai_select <ai>)

selects the specified squad.

(ai_deselect)

clears the selected encounter.

(ai_set_deaf <ai> <boolean>)

enables or disables hearing for actors in the specified encounter.

(ai_set_blind <ai> <boolean>)

enables or disables sight for actors in the specified encounter.

(ai_magically_see <ai> <ai>)

Make one squad magically aware of another.

(ai_magically_see_object <ai> <object>)

Make a squad magically aware of a particular object.

(ai_timer_start <ai>)

makes a squad's delay timer start counting.

(ai_timer_expire <ai>)

makes a squad's delay timer expire and releases them to enter combat.

(ai_migrate <ai> <ai>)

makes all or part of an encounter move to another encounter.

(ai_allegiance <team> <team>)

creates an allegiance between two teams.

(ai_allegiance_remove <team> <team>)

destroys an allegiance between two teams.

(ai_braindead <ai> <boolean>)

makes a group of actors braindead, or restores them to life (in their initial state)

(ai_braindead_by_unit <object_list> <boolean>)

makes a list of objects braindead, or restores them to life. if you pass in a vehicle index, it makes all actors in that vehicle braindead (including any built-in guns)

(ai_disregard <object_list> <boolean>)

if TRUE, forces all actors to completely disregard the specified units, otherwise lets them acknowledge the units again

(ai_prefer_target <object_list> <boolean>)

if TRUE, *ALL* enemies will prefer to attack the specified units. if FALSE, removes the preference.

(ai_teleport_to_starting_location <ai>)

teleports a group of actors to the starting locations of their current squad(s)

(ai_teleport_to_starting_location_if_unsupported <ai>)

teleports a group of actors to the starting locations of their current squad(s), only if they are not supported by solid ground (i.e. if they are falling after switching BSPs)

(ai_renew <ai>)

refreshes the health and grenade count of a group of actors, so they are as good as new

(ai_force_active <ai> <boolean>)

forces an encounter to remain active (i.e. not freeze in place) even if there are no players nearby

(ai_force_active_by_unit <unit> <boolean>)

forces a named actor that is NOT in an encounter to remain active (i.e. not freeze in place) even if there are no players nearby

(ai_playfight <ai> <boolean>)

sets an encounter to be playfighting or not

(ai_reconnect)

reconnects all AI information to the current structure bsp (use this after you create encounters or command lists in sapien, or place new firing points or command list points)

(ai_vehicle_enterable_distance <unit> <real>)

sets a vehicle as being impulsively enterable for actors within a certain distance

(ai_vehicle_enterable_team <unit> <team>)

sets a vehicle as being impulsively enterable for actors on a certain team

(ai_vehicle_enterable_actor_type <unit> <actor_type>)

sets a vehicle as being impulsively enterable for actors of a certain type (grunt, elite, marine etc)

(ai_vehicle_enterable_actors <unit> <ai>)

sets a vehicle as being impulsively enterable for a certain encounter/squad of actors

(ai_vehicle_enterable_disable <unit>)

disables actors from impulsively getting into a vehicle (this is the default state for newly placed vehicles)

(ai_look_at_object <unit> <object>)

tells an actor to look at an object until further notice

(ai_stop_looking <unit>)

tells an actor to stop looking at whatever it's looking at

(ai_automatic_migration_target <ai> <boolean>)

enables or disables a squad as being an automatic migration target

(ai_berserk <ai> <boolean>)

forces a group of actors to start or stop berserking

(ai_set_team <ai> <team>)

makes an encounter change to a new team

(ai_allow_dormant <ai> <boolean>)

either enables or disables automatic dormancy for a group of actors

(ai_is_attacking <ai>)

returns whether a platoon is in the attacking mode (or if an encounter is specified, returns whether any platoon in that encounter is attacking)

(ai_fighting_count <ai>)

return the number of actors that are fighting in a squad or squad_group

(ai_living_count <ai>)

return the number of living actors in the specified encounter and/or squad.

(ai_living_fraction <ai>)

return the fraction [0-1] of living actors in the specified encounter and/or squad.

(ai_strength <ai>)

return the current strength (average body vitality from 0-1) of the specified encounter and/or squad.

(ai_swarm_count <ai>)

return the number of swarm actors in the specified encounter and/or squad.

(ai_nonswarm_count <ai>)

return the number of non-swarm actors in the specified encounter and/or squad.

(ai_actors <ai>)

converts an ai reference to an object list.

(ai_allegiance_broken <team> <team>)

returns whether two teams have an allegiance that is currently broken by traitorous behavior

(ai_set_orders <ai> <ai_orders>)

Takes the squad or squad group (arg1) and gives it the order (arg3) in zone (arg2). Use the zone_name/order_name format

(ai_spawn_count <ai>)

returns the number of actors spawned in the given squad or squad group

(ai_trigger_test <string> <ai>)

Tests the named trigger on the named squad

(generate_pathfinding)

Generate pathfinding info for all structure bsps in the current scenario

(ai_sectors_intersect <long> <long>)

Intersect two sectors (for debugging)

(ai_sector_link_between <long> <long>)

Returns the link that connects the two given sectors

(ai_render_paths_all)

Turns on raw, smoothed, avoided paths and avoidance obstacles

(ai_vehicle_get <ai>)

Returns the vehicle that the given actor is in.

(ai_vehicle_get_from_starting_location <ai>)

Returns the vehicle that was spawned at the given starting location.

(ai_vehicle_reserve_seat <vehicle> <string> <boolean>)

Reserves the given seat on the given vehicle (so that AI may not enter it

(ai_vehicle_reserve <vehicle> <boolean>)

Reserves the given vehicle (so that AI may not enter it

(ai_vehicle_enter <ai> <unit> <string>)

tells a group of actors to get into a vehicle, in the substring-specified seats (e.g. passenger for pelican)... does not interrupt any actors who are already going to vehicles

(ai_vehicle_enter <ai> <unit>)

tells a group of actors to get into a vehicle... does not interrupt any actors who are already going to vehicles

(ai_vehicle_enter_immediate <ai> <unit> <string>)

the given group of actors is snapped into a vehicle, in the substring-specified seats (e.g. passenger for pelican)... does not interrupt any actors who are already going to vehicles

(ai_vehicle_enter_immediate <ai> <unit>)

the given group of actors is snapped into a vehicle

(ai_vehicle_exit <ai> <string>)

tells a group of actors to get out of any vehicles that they are in (if their seat matches the substring)

(ai_vehicle_exit <ai>)

tells a group of actors to get out of any vehicles that they are in

(vehicle_overturned <vehicle>)

Returns true if the vehicle is overturned

(vehicle_flip <vehicle>)

Flips an overturned vehicle

(ai_combat_status <ai>)

Returns the highest integer combat status in the given squad-group/squad/actor

(ai_verify_tags)

Verifies state of ai-related tags (e.g. orders, squads, zones, etc.)

(ai_wall_lean <ai>)

Makes the actor lean against a wall RIGHT NOW

(cs_run_command_script <ai> <ai_command_script>)

Causes the specified actor(s) to start executing a command script immediately (discarding any other command scripts in the queue)

(cs_queue_command_script <ai> <ai_command_script>)

Add a command script onto the end of an actor's command script queue

(cs_stack_command_script <ai> <ai_command_script>)

Push a command script to the top of the actor's command script queue

(cs_run_joint_command_script <ai_command_script> <ai> <ai>)

Causes the specified actor(s) to start executing a command script immediately (discarding any other command scripts in the queue)

(cs_run_joint_command_script <ai_command_script> <ai> <ai> <ai>)

Causes the specified actor(s) to start executing a command script immediately (discarding any other command scripts in the queue)

(cs_command_script_running <ai> <ai_command_script>)

Returns true if the ai is running the command script in question

(cs_command_script_queued <ai> <ai_command_script>)

Returns true if the command script is in the ai's cs queue

(cs_number_queued <ai>)

Returns the number of queued command scripts

(cs_switch <short>)

Switch control of the joint command script to the given member

(cs_transfer <ai>)

Transfer control of the command script to the given actor (replacing what he has)

(cs_transfer_stack <ai>)

Transfer control of the command script to the given actor (stacking it)

(cs_transfer_queue <ai>)

Transfer control of the command script to the given actor (queueing it)

(cs_fly_to <point reference>)

Flies the actor to the given point

(cs_fly_to <point reference> <real>)

Flies the actor to the given point (within the given tolerance)

(cs_fly_to_and_face <point reference> <point reference>)

Flies the actor to the given point and orients him in the appropriate direction

(cs_fly_to_and_face <point reference> <point reference> <real>)

Flies the actor to the given point and orients him in the appropriate direction (within the given tolerance)

(cs_fly_by <point reference>)

Flies the actor through the given point

(cs_fly_by <point reference> <real>)

Flies the actor through the given point (within the given tolerance)

(cs_go_to <point reference>)

Moves the actor to a specified point

(cs_go_to <point reference> <real>)

Moves the actor to a specified point, attenuating throttle by the given amount when near the goal

(cs_go_by <point reference> <point reference>)

Actor moves toward the point, and considers it hit when it breaks the indicated plane

(cs_go_by <point reference> <point reference> <real>)

Actor moves toward the point, and considers it hit when it breaks the indicated plane, attenuating throttle by the given amount when near the goal

(cs_go_to_and_face <point reference> <point reference>)

Moves the actor to a specified point and has him face the second point

(cs_look <boolean> <point reference>)

Actor looks at the point for the remainder of the cs, or until overridden

(cs_look_player <boolean>)

Actor looks at nearest player for the duration of the cs, or until overridden

(cs_look_object <boolean> <object>)

Actor looks at the object for the duration of the cs, or until overridden

(cs_aim <boolean> <point reference>)

Actor aims at the point for the remainder of the cs, or until overridden (overrides look)

(cs_aim_player <boolean>)

Actor aims at nearest player for the duration of the cs, or until overridden (overrides look)

(cs_aim_object <boolean> <object>)

Actor aims at the object for the duration of the cs, or until overridden (overrides look)

(cs_face <boolean> <point reference>)

Actor faces exactly the point for the remainder of the cs, or until overridden (overrides aim, look)

(cs_face_player <boolean>)

Actor faces exactly the nearest player for the duration of the cs, or until overridden (overrides aim, look)

(cs_face_object <boolean> <object>)

Actor faces exactly the given object for the duration of the cs, or until overridden (overrides aim, look)

(cs_move_in_direction <real> <real> <real>)

Actor moves at given angle, for the given distance, optionally with the given facing (angle, distance, facing)

(cs_pause <real>)

The actor does nothing for the given number of seconds

(cs_shoot <boolean>)

Actor is allowed to shoot at its target or not

(cs_shoot <boolean> <object>)

Actor shoots at given target

(cs_shoot_point <boolean> <point reference>)

Actor shoots at given point

(cs_vehicle_speed <real>)

Set the speed at which the actor will drive a vehicle, expressed as a multiplier 0-1

(cs_grenade <point reference> <short>)

Actor throws a grenade, either by tossing (arg2=0), lobbing (1) or bouncing (2)

(cs_jump <real> <real>)

Actor jumps in direction of angle at the given velocity (angle, velocity)

(cs_jump_to_point <real> <real>)

Actor jumps with given horizontal and vertical velocity

(cs_vocalize <short>)

Actor emits vocalization of given type

(cs_play_sound <sound>)

Actor plays an impulse sound

(cs_play_sound <sound> <real>)

Actor plays an impulse sound and the atom blocks for the given percentage of the sound's total length

(cs_play_sound <sound> <real> <real>)

Actor plays an impulse sound and the atom blocks for the given percentage of the sound's total length, at the given volume (0..1)

(cs_stop_sound <sound>)

Stops the specified impulse sound.

(cs_custom_animation <animation_graph> <string> <real> <boolean>)

starts a custom animation playing on the unit (interpolates into animation if last parameter is TRUE)

(cs_stop_custom_animation)

Stop running a custom animation

(cs_die <short>)

Actor dies in specified manner

(cs_teleport <point reference> <point reference>)

Actor teleports to point1 facing point2

(cs_animate <long> <short>)

Actor performs animation with given modifier (anim-ref, modifier)

(cs_animation_mode <short>)

Actor switches to given animation mode

(cs_crouch <boolean>)

Actor crouches for the remainder of the command script, or until overridden

(cs_set_pathfinding_radius <real>)

Sets the actor's pathfinding radius (this distance at which a destination is considered to have been reached) for the remainder of the command script

(cs_go_to_vehicle <vehicle>)

Actor gets in the appropriate vehicle

(cs_set_behavior <ai_behavior>)

Actor performs the indicated behavior

(cs_formation <short> <ai> <point reference> <point reference>)

Actor initiates a formation of the given type at the given point, facing (initially) at the given other point. Formation types are (0) 1x column (1) 2x column ... (4) wall (5) wedge.

(cs_ignore_obstacles <boolean>)

Actor does not avoid obstacles when true

(cs_turn_sharpness <boolean> <real>)

Set the sharpness of a vehicle turn (values 0 -> 1). Only applicable to nondirectional flying vehicles (e.g. dropships)

(cs_vehicle_speed_instantaneous <real>)

Set the instantaneous speed of the vehicle we're driving

(cs_abort_on_alert <boolean>)

Command script ends prematurely when actor's combat status raises to 'alert' or higher

(cs_abort_on_damage <boolean>)

Command script ends prematurely when actor is damaged

(cs_enable_targeting <boolean>)

Actor autonomous target selection disabled.

(cs_enable_looking <boolean>)

Actor autonomous looking disabled.

(cs_enable_moving <boolean>)

Actor autonomous moving disabled.

(cs_set_style <style>)

Override the actor's style

(cs_force_combat_status <short>)

Force the actor's combat status (0= no override, 1= asleep, 2=idle, 3= alert, 4= active)

(cs_enable_pathfinding_failsafe <boolean>)

Actor blocks until pathfinding calls succeed

(camera_control <boolean>)

toggles script control of the camera.

(camera_set <cutscene_camera_point> <short>)

moves the camera to the specified camera point over the specified number of ticks.

(camera_set_relative <cutscene_camera_point> <short> <object>)

moves the camera to the specified camera point over the specified number of ticks (position is relative to the specified object).

(camera_set_animation <animation_graph> <string>)

begins a prerecorded camera animation.

(camera_set_animation_relative <animation_graph> <string> <unit> <cutscene_flag>)

begins a prerecorded camera animation synchronized to unit relative to cutscene flag.

(camera_set_first_person <unit>)

makes the scripted camera follow a unit.

(camera_set_dead <unit>)

makes the scripted camera zoom out around a unit as if it were dead.

(camera_place_relative <object>)

all subsequent camera placement in sapien be marked as relative to this object

(camera_place_worldspace)

all subsequent camera placement in sapien will be marked as worldspace

(camera_time)

returns the number of ticks remaining in the current camera interpolation.

(camera_set_field_of_view <real>)

sets the field of view

(debug_camera_save)

saves the camera position and facing.

(debug_camera_load)

loads the saved camera position and facing.

(debug_camera_save_name <string>)

saves the camera position and facing to filename

(debug_camera_load_name <string>)

loads the camera position and facing from filename

(director_debug_camera <boolean>)

enable/disable camera debugging

(camera_set_pan <cutscene_camera_point> <short>)

moves the camera to the specified camera point over the specified number of ticks with a constant speed.

(camera_pan <cutscene_camera_point> <cutscene_camera_point> <short> <short> <real> <short> <real>)

camera_pan <start point> <end point> <ticks> <ease-in ticks> <start velocity scale> <ease-out ticks> <end velocity scale>

(camera_predict <cutscene_camera_point>)

predictes resources given a camera view

(game_time)

gets ticks elapsed since the start of the game.

(game_difficulty_get)

returns the current difficulty setting, but lies to you and will never return easy, instead returning normal

(game_difficulty_get_real)

returns the actual current difficulty setting without lying

(pvs_set_object <object>)

sets the specified object as the special place that activates everything it sees.

(pvs_set_camera <cutscene_camera_point>)

sets the specified cutscene camera point as the special place that activates everything it sees.

(pvs_clear)

removes the special place that activates everything it sees.

(players_unzoom_all)

resets zoom levels on all players

(player_enable_input <boolean>)

toggle player input. the player can still free-look, but nothing else.

(player_camera_control <boolean>)

enables/disables camera control globally

(player_action_test_reset)

resets the player action test state so that all tests will return false.

(player_action_test_jump)

returns true if any player has jumped since the last call to (player_action_test_reset).

(player_action_test_primary_trigger)

returns true if any player has used primary trigger since the last call to (player_action_test_reset).

(player_action_test_grenade_trigger)

returns true if any player has used grenade trigger since the last call to (player_action_test_reset).

(player_action_test_zoom)

returns true if any player has hit the zoom button since the last call to (player_action_test_reset).

(player_action_test_rotate_weapons)

returns true if any player has hit the rotate-weapon button since the last call to (player_action_test_reset).

(player_action_test_rotate_grenades)

returns true if any player has hit the rotate-grenades button since the last call to (player_action_test_reset).

(player_action_test_action)

returns true if any player has hit the action key since the last call to (player_action_test_reset).

(player_action_test_accept)

returns true if any player has hit accept since the last call to (player_action_test_reset).

(player_action_test_cancel)

returns true if any player has hit cancel key since the last call to (player_action_test_reset).

(player_action_test_look_relative_up)

returns true if any player has looked up since the last call to (player_action_test_reset).

(player_action_test_look_relative_down)

returns true if any player has looked down since the last call to (player_action_test_reset).

(player_action_test_look_relative_left)

returns true if any player has looked left since the last call to (player_action_test_reset).

(player_action_test_look_relative_right)

returns true if any player has looked right since the last call to (player_action_test_reset).

(player_action_test_look_relative_all_directions)

returns true if any player has looked up, down, left, and right since the last call to (player_action_test_reset).

(player_action_test_move_relative_all_directions)

returns true if any player has moved forward, backward, left, and right since the last call to (player_action_test_reset).

(player_action_test_start)

returns true if any player has pressed the start button since the last call to (player_action_test_reset).

(player_action_test_back)

returns true if any player has pressed the back button since the last call to (player_action_test_reset).

(debug_teleport_player <long> <long>)

for testing: teleports one player to another's location

(map_reset)

starts the map from the beginning.

(switch_bsp <short>)

takes off your condom and changes to a different structure bsp

(switch_bsp_by_name <string>)

leaves your condom on and changes to a different structure bsp by name

(structure_bsp_index)

returns the current structure bsp index

(crash <string>)

crashes (for debugging).

(version)

prints the build version.

(status)

prints the value of all global status variables.

(record_movie <short> <long> <short>)

<frame rate> <seconds> <screen size>

(record_movie_distributed <short> <long> <short> <long> <long>)

<frame rate> <seconds> <screen size> <mod count> <mod index>

(screenshot <string>)

takes a screenshot and saves as <name>.tif

(screenshot_big <short> <string>)

takes an NxN multiple-page screenshot and saves as <name>.tif

(screenshot_big_jittered <short> <string>)

takes an NxN subpixel sampled 640x480 screenshot and saves as <name>.tif

(screenshot_cubemap <string>)

takes a cubemap screenshot and saves as <name>.tif

(main_menu)

goes back to the main menu

(main_halt)

goes to a halted pregame state

(map_name <string>)

the same as game_start: launches a game for debugging purposes

(game_multiplayer <string>)

debug map launching: sets the multiplayer variant for the next map.

(game_splitscreen <long>)

debug map launching: sets the number of multiplayer splitscreen players for the next map.

(game_difficulty <game_difficulty>)

debug map launching: sets the difficulty of the next map.

(game_coop_players <long>)

debug map launching: sets the number of coop players for the next map.

(game_initial_bsp <long>)

debug map launching: sets the initial bsp for the next map.

(game_start <string>)

debug map launching: starts a game on the specified map.

(texture_cache_flush)

(geometry_cache_flush)

(sound_cache_flush)

(debug_memory)

dumps memory leaks.

(debug_memory_by_file)

dumps memory leaks by source file.

(debug_memory_for_file <string>)

dumps memory leaks from the specified source file.

(debug_tags)

writes all memory being used by tag files into tag_dump.txt

(tags_verify_all)

verifies usage of infidel fields is correct

(profile_reset)

resets profiling data.

(profile_activate <string>)

activates profile sections based on a substring.

(profile_deactivate <string>)

deactivates profile sections based on a substring.

(profile_mode <string>)

sets the current profiling mode.

(damage_control_get <string>)

gets a damage control setting by string

(damage_control_set <string> <boolean>)

sets a damage control setting by string

(ai_lines)

cycl## es through AI line-spray modes

(ai_debug_sound_point_set)

drops the AI debugging sound point at the camera location

(ai_debug_vocalize <string> <string>)

makes the selected AI vocalize

(ai_debug_teleport_to <string>)

teleports all players to the specified zone

(ai_debug_speak <string>)

makes the currently selected AI speak a vocalization (e.g. ai_speak "pain minor")

(ai_debug_speak_list <string>)

makes the currently selected AI speak a list of vocalizations (e.g. ai_speak_list "involuntary")

(fade_in <real> <real> <real> <short>)

does a screen fade in from a particular color

(fade_out <real> <real> <real> <short>)

does a screen fade out to a particular color

(cinematic_start)

initializes game to start a cinematic (interruptive) cutscene

(cinematic_stop)

initializes the game to end a cinematic (interruptive) cutscene

(cinematic_skip_start_internal)

(cinematic_skip_stop_internal)

(cinematic_show_letterbox <boolean>)

sets or removes the letterbox bars

(cinematic_set_title <cutscene_title>)

activates the chapter title

(cinematic_set_title_delayed <cutscene_title> <real>)

activates the chapter title, delayed by <real> seconds

(cinematic_suppress_bsp_object_creation <boolean>)

suppresses or enables the automatic creation of objects during cutscenes due to a bsp switch

(cinematic_dolly_start <object> <string>)

lock dolly-zoom onto object marker

(cinematic_dolly_stop <real>)

unlock dolly-zoom <seconds>

(cinematic_subtitle <string> <real>)

lock dolly-zoom onto object marker

(attract_mode_start)

starts an attract mode movie

(game_won)

causes the player to successfully finish the current level and move to the next

(game_lost)

causes the player to revert to his previous saved game

(game_is_cooperative)

returns TRUE if the game is cooperative

(game_skip_ticks <long>)

skips some amount of game ticks. only usable in cutscenes, for durations less than 1/2 second.

(game_revert)

reverts to last saved game, if any (for testing, the first bastard that does this to me gets it in the head)

(game_save_unsafe)

saves right now, even if the game is in an immediate-loss state (NEVER USE THIS! EVER!)

(debug_spawning <string> <boolean>)

debugs spawn points for the inputted player

(king_set_hill <long>)

sets which index the active hill should be

(headhunter_set_bin <long>)

sets which index the active hill should be

(core_load)

loads debug game state from core\core.bin

(core_load_name <string>)

loads debug game state from core\<path>

(core_save)

saves debug game state to core\core.bin

(core_save_name <string>)

saves debug game state to core\<path>

(core_load_game)

loads level and game state from core\core.bin

(core_load_game_name <string>)

loads level and game state from core\<path>

(core_regular_upload_to_debug_server <boolean>)

toggle periodic core uploading to debug server

(core_upload)

saves the game state to disk core\core.bin and uploads it to the debug server

(core_upload_name <string>)

saves the game state to disk core\<path> and uploads it to the debug server

(core_set_upload_option <string>)

sets options for game state uploading (current options are 'default', 'repro', and 'stress'

(game_safe_to_save)

returns FALSE if it would be a bad idea to save the player's game right now

(game_all_quiet)

returns FALSE if there are bad guys around, projectiles in the air, etc.

(game_safe_to_speak)

returns FALSE if it would be a bad idea to save the player's game right now

(game_save)

checks to see if it is safe to save game, then saves (gives up after 8 seconds)

(game_save_cancel)

cancels any pending game_save, timeout or not

(game_save_no_timeout)

checks to see if it is safe to save game, then saves (this version never gives up)

(game_save_immediate)

disregards player's current situation and saves (BE VERY CAREFUL!)

(game_saving)

checks to see if the game is trying to save the map.

(game_reverted)

(sound_impulse_predict <sound>)

(sound_impulse_trigger <sound> <object> <real> <long>)

plays an impulse sound from the specified source object (or "none"), with the specified scale.

(sound_impulse_start <sound> <object> <real>)

plays an impulse sound from the specified source object (or "none"), with the specified scale.

(sound_impulse_start_cinematic <sound> <object> <real> <real> <real>)

<sound> <object> <scale> <3d gain> <first person gain> plays an impulse sound from the specified source object.

(sound_impulse_start_effect <sound> <object> <real> <string>)

plays an impulse sound from the specified source object (or "none"), with the specified scale and effect.

(sound_impulse_time <sound>)

returns the time remaining for the specified impulse sound.

(sound_impulse_stop <sound>)

stops the specified impulse sound.

(sound_looping_predict <looping_sound>)

(sound_looping_start <looping_sound> <object> <real>)

plays a looping sound from the specified source object (or "none"), with the specified scale.

(sound_looping_stop <looping_sound>)

stops the specified looping sound.

(sound_looping_stop_immediately <looping_sound>)

stops the specified looping sound immediately.

(sound_looping_set_scale <looping_sound> <real>)

changes the scale of the sound (which should affect the volume) within the range 0 to 1.

(sound_looping_set_alternate <looping_sound> <boolean>)

enables or disables the alternate loop/alternate end for a looping sound.

(sound_loop_spam)

start all loaded looping sounds

(sound_class_show_channel <string> <boolean>)

shows/hides sound classes w/ substring in debug_sound_channels view

(sound_class_debug_sound_start <string> <boolean>)

shows/hides when sounds of sound classes w/ substring start

(debug_sounds_enable <string> <boolean>)

enables or disables all sound classes matching the substring.

(sound_class_set_gain <string> <real> <short>)

changes the gain on the specified sound class(es) to the specified gain over the specified number of ticks.

(sound_class_set_gain_db <string> <real> <short>)

changes the gain on the specified sound class(es) to the specified gain(dB) over the specified number of ticks.

(sound_class_enable_ducker <string> <boolean>)

enables or disables the ducker on all sound classes matching the substring.

(debug_sound_environment_parameter <long> <real>)

(sound_set_global_effect <string> <real>)

(sound_set_global_effect_scale <string> <real>)

(vehicle_hover <vehicle> <boolean>)

stops the vehicle from running real physics and runs fake hovering physics instead.

(vehicle_count_bipeds_killed <vehicle>)

returns how many people this vehicle has killed

(show_hud <boolean>)

shows or hides the hud

(show_hud_help_text <boolean>)

shows or hides the hud help text

(show_hud_messages <boolean>)

shows or hides the hud messages

(enable_hud_help_flash <boolean>)

starts/stops the help text flashing

(hud_help_flash_restart)

resets the timer for the help text flashing

(activate_nav_point_flag <navpoint> <unit> <cutscene_flag> <real>)

activates a nav point type <string> attached to (local) player <unit> anchored to a flag with a vertical offset <real>. If the player is not local to the machine, this will fail

(activate_nav_point_object <navpoint> <unit> <object> <real>)

activates a nav point type <string> attached to (local) player <unit> anchored to an object with a vertical offset <real>. If the player is not local to the machine, this will fail

(activate_team_nav_point_flag <navpoint> <team> <cutscene_flag> <real>)

activates a nav point type <string> attached to a team anchored to a flag with a vertical offset <real>. If the player is not local to the machine, this will fail

(activate_team_nav_point_object <navpoint> <team> <object> <real>)

activates a nav point type <string> attached to a team anchored to an object with a vertical offset <real>. If the player is not local to the machine, this will fail

(deactivate_nav_point_flag <unit> <cutscene_flag>)

deactivates a nav point type attached to a player <unit> anchored to a flag

(deactivate_nav_point_object <unit> <object>)

deactivates a nav point type attached to a player <unit> anchored to an object

(deactivate_team_nav_point_flag <team> <cutscene_flag>)

deactivates a nav point type attached to a team anchored to a flag

(deactivate_team_nav_point_object <team> <object>)

deactivates a nav point type attached to a team anchored to an object

(cls)

clears console text from the screen

(error_overflow_suppression <boolean>)

enables or disables the suppression of error spamming

(error_geometry_show <string>)

highlights all error geometry with a name that includes the given substring

(error_geometry_hide <string>)

hides all error geometry with a name that includes the given substring

(error_geometry_show_all)

highlights all error geometry

(error_geometry_hide_all)

hides all error geometry

(error_geometry_list)

prints out a list of all error geometry types and counts

(player_effect_set_max_translation <real> <real> <real>)

<x> <y> <z>

(player_effect_set_max_rotation <real> <real> <real>)

<yaw> <pitch> <roll>

(player_effect_set_max_rumble <real> <real>)

<left> <right>

(player_effect_start <real> <real>)

<max_intensity> <attack time>

(player_effect_stop <real>)

<decay>

(hud_show_health <boolean>)

hides/shows the health panel

(hud_blink_health <boolean>)

starts/stops manual blinking of the health panel

(hud_show_shield <boolean>)

hides/shows the shield panel

(hud_blink_shield <boolean>)

starts/stops manual blinking of the shield panel

(hud_show_motion_sensor <boolean>)

hides/shows the motion sensor panel

(hud_blink_motion_sensor <boolean>)

starts/stops manual blinking of the motion sensor panel

(hud_show_crosshair <boolean>)

hides/shows the weapon crosshair

(hud_show_ammo <boolean>)

hides/shows the weapon/grenade ammo counter

(hud_clear_messages)

clears all non-state messages on the hud

(hud_set_help_text <hud_message>)

displays <message> as the help text

(hud_set_objective_text <hud_message>)

sets <message> as the current objective

(hud_set_timer_time <short> <short>)

sets the time for the timer to <short> minutes and <short> seconds, and starts and displays timer

(hud_set_timer_warning_time <short> <short>)

sets the warning time for the timer to <short> minutes and <short> seconds

(hud_set_timer_position <short> <short> <hud_corner>)

sets the timer upper left position to (x, y)=>(<short>, <short>)

(show_hud_timer <boolean>)

displays the hud timer

(pause_hud_timer <boolean>)

pauses or unpauses the hud timer

(hud_get_timer_ticks)

returns the ticks left on the hud timer

(time_code_show <boolean>)

shows the time code timer

(time_code_start <boolean>)

starts/stops the time code timer

(time_code_reset)

resets the time code timer

(rasterizer_profile_include <string>)

(rasterizer_profile_include_all)

(rasterizer_profile_include_all_except <string>)

(rasterizer_profile_exclude <string>)

(rasterizer_profile_exclude_all)

(rasterizer_profile_exclude_all_except <string>)

(rasterizer_artist_profile <long>)

used to turn on artist profile modes 0-disabled, 1-simple mode, 2-detailed mode

(rasterizer_debug_display_bitmap <string>)

displays a bitmap

(rasterizer_decals_flush)

flush all decals

(rasterizer_lens_flares_reset_for_new_map)

(decorator_debug_set_filter <long> <long> <long>)

(decorator_debug_precision_add)

(decorator_debug_density_erase)

(decorator_debug_density_low)

(decorator_debug_density_high)

(script_screen_effect_set_value <short> <real>)

sets a screen effect script value

(cinematic_screen_effect_start <boolean>)

starts screen effect pass TRUE to clear

(cinematic_screen_effect_set_convolution <short> <short> <real> <real> <real>)

sets the convolution effect

(cinematic_screen_effect_set_filter <real> <real> <real> <real> <boolean> <real>)

sets the filter effect

(cinematic_screen_effect_set_filter_desaturation_tint <real> <real> <real>)

sets the desaturation filter tint color

(cinematic_screen_effect_set_video <short> <real>)

sets the video effect: <noise intensity[0,1]>, <overbright: 0=none, 1=2x, 2=4x>

(cinematic_screen_effect_set_depth_of_field <real> <real> <real> <real> <real> <real> <real>)

sets dof: <seperation dist>, <near blur lower bound> <upper bound> <time> <far blur lower bound> <upper bound> <time>

(cinematic_screen_effect_set_bloom <real> <real> <real> <real> <real>)

blur threshold overbright tint modulation

(bloom <real> <real> <real>)

new brightness, new overbrightness, delta time

(cinematic_bloom <real> <real> <real>)

new brightness, new overbrightness, delta time

(cinematic_screen_effect_set_bloom_transition <real> <real> <real> <real> <real> <real>)

blur threshold overbright tint modulation transition-time

(cinematic_screen_effect_set_bloom_threshold_color <real> <real> <real>)

red green blue

(cinematic_screen_effect_set_bloom_overbright_color <real> <real> <real>)

red green blue

(cinematic_screen_effect_set_bloom_tint_color <real> <real> <real>)

red green blue

(cinematic_screen_effect_set_bloom_modulation_color <real> <real> <real>)

red green blue

(cinematic_screen_effect_set_crossfade <real>)

transition-time

(cinematic_screen_effect_set_crossfade2 <real> <real>)

transition-time, exponent

(cinematic_screen_effect_stop)

returns control of the screen effects to the rest of the game

(cinematic_set_near_clip_distance <real>)

(cinematic_set_far_clip_distance <real>)

(cinematic_set_environment_map_attenuation <real> <real> <real>)

interpolates environment-map attenuation (on flagged shaders) from <low> to <high> over <time>

(cinematic_set_environment_map_bitmap <bitmap>)

sets environment-map bitmap (on flagged shaders) instantly

(cinematic_reset_environment_map_bitmap)

resets environment-map bitmap (on flagged shaders) to default instantly

(cinematic_set_environment_map_tint <real> <real> <real> <real> <real> <real> <real> <real>)

perpendicular color: (red green blue brightness), parallel color: (red green blue brightness)... sets environment-map tint (on flagged shaders) instantly

(cinematic_reset_environment_map_tint)

resets environment-map tint (on flagged shaders) to default instantly

(cinematic_layer <long> <real> <real>)

interpolates the value of <cinematic layer x> from current position to <value> over <time>

(cinematic_dynamic_reflections <boolean> <boolean>)

sets up dynamic reflections: <enabled: [true, false]> <filtering enabled: [true, false]>

(lightmaps_expose <real> <real> <real>)

re-exposes the lightmap palettes

(create_player_profile <string> <short> <short> <boolean>)

create a player profile

(controller_invert_look)

invert look on all attached controllers

(controller_look_speed <short>)

set look speed for all attached controllers

(ui_debug_load_main_menu)

loads the main menu screen

(ui_debug_text_bounds <boolean>)

toggle rendering of widget text boundaries

(ui_debug_show_title_safe_bounds <boolean>)

toggle display of title safe boundary

(ui_dispose)

dispose of halo2 gameshell ui

(ui_debug_screen_tag <string>)

test a ui screen

(ui_transition_out_console_window)

transition out any ui on the console window

(ui_debug_show_screen_tag_path <boolean>)

display tag path of screens as they load

(ui_set_beta <boolean>)

set ui beta testing on/off

(ui_set_automation_mode <short>)

set ui automation mode

(ui_set_automation_hopper_type <short>)

set ui / mp automation hopper

(ui_set_automation_desired_local_user_count <short>)

set ui / mp automation desired local user count

(ui_set_automation_desired_desired_network_game_player_count <short>)

set ui / mp automation desired network game player count

(ui_set_automation_desired_network_game_length_seconds <long>)

set ui / mp automation desired game time length

(ui_set_automation_desired_network_session_name <string>)

set ui / mp automation desired session name

(ui_set_automation_desired_controller_team <short> <short>)

set desired mp team for a controller

(ui_set_automation_desired_controller_player_profile <short> <string> <string> <boolean>)

set desired player profile and gamertag for a controller

(input_suppress_rumble <boolean>)

disable the friggin' rumble

(update_remote_camera)

force synchronization of remote machine camera

(net_status_filter <string>)

filters the set of network status to display

(net_sim_reset)

network simulation: resets the simulation state

(net_sim_spike_now)

network simulation: starts a latency spike immediately

(net_sim_dropspike_now)

network simulation: starts a packet loss spike immediately

(net_test_ping)

network test: sends a ping

(net_test_channel_loopback)

network test: creates loopback channels

(net_test_channel_delete)

network test: deletes all channels

(net_test_leave_squad)

network test: leave current squad

(net_test_delegate_leader <long>)

network test: delegate leadership to the specified player

(net_test_map_name <string>)

network test: sets the name of the scenario to play

(net_test_player_color <long>)

network test: temporarily sets the color for all local players

(net_test_reset_objects)

network test: resets all objects on the map

(net_set_machine_name <string>)

sets the nickname of your xbox

(net_event_display_category <string> <network_event>)

sets the display level for a named category of network events

(net_event_log_category <string> <network_event>)

sets the log level for a named category of network events

(net_event_list_categories <string>)

lists all categories that exist under a particular category string

(net_test_create_group_session)

network test: create group session

(net_test_matchmaking_initiate)

network test: initiate matchmaking

(net_test_matchmaking_abort)

network test: abort the matchmaking process

(play_bink_movie <string>)

(set_global_doppler_factor <real>)

new doppler factor: <real>

(set_global_mixbin_headroom <long> <long>)

(data_mine_insert <string>)

insert text and camera position in the data mine

(data_mine_upload)

upload all data mining data files to debug server

(data_mine_playback <string>)

loads and displays data mine data from a file

(data_mine_playback_last)

loads and displays last data mine data

(data_mine_playback_exit)

exit data visualization

(data_mine_enable <boolean>)

enable/disable data mining

(data_mine_track_event <string>)

enable mining of an event

(data_mine_display_event <string>)

enable displaying of an event

(data_mine_show_all_events)

show all data mine events

(data_mine_show_tracked_events)

show what events are being tracked by the data mine

(data_mine_show_displayed_events)

show what events are being displayed by the data mine

(data_mine_display_session_data)

show data mine session, game, and network ids

(data_mine_display_disk_writes <boolean>)

enable/disable console message on disk writes

(error_enable <string> <boolean>)

enables/disables display for a class of errors

(render_layer_enable <string> <boolean>)

enable/disables a render_layer

(render_layer_enable_all <boolean>)

enable/disables all render_layers

(rasterizer_overdraw_z)

toggles overdraw with z compare on

(rasterizer_overdraw)

toggles overdraw with z compare off

(test_memory_allocators <long> <short> <short> <short> <long>)

performs tests on different memory allocators

(test_memory_allocators <long> <short> <short> <short> <long> <string>)

performs tests on different memory allocators and saves the results

(test_xcr_monkey_enable <boolean>)

enable/disable controller monkeys for all in game players

(test_web_map_snapshot <string>)

takes two special screenshots and saves them, along with the camera information, as <name>.tif, <name>_secondary.tif and <name>_camera.txt

(test_telnet_status_enable <boolean>)

enable/disable status events being sent to the telnet console

(test_telnet_status_interval <long>)

sets the interval that status events are sent to the telnet console.

(script_temporary_disable_lightmap_shadows <boolean>)

disable lightmap shadows

(flag_new <string> <string>)

<name> <description>

(flag_new_at_look <string> <string>)

<name> <description>

(flags_clear)

erases all comment flags when not in editor (sapien)

(flags_save)

dump comment flags to vrml file

(flags_save_filtered <string>)

<substring filter>

(flags_save_named <string>)

<file name>

(flags_save_named_filtered <string> <string>)

<filter string> <file name>

(flags_default_name <string>)

<default comment flag name>

(flags_default_comment <string>)

<default comment flag description>

(flags_set_filter <string>)

<flag name filter>

(flags_export)

dump comment flags to a .txt file

(flags_export_filtered <string>)

<substring filter>

(flags_export_named <string>)

<file name>

(flags_export_named_filtered <string> <string>)

<filter string> <file name>

Halo 2 Anniversary Mod Documentation Coming Soon

11/23/2022 • 2 minutes to read

Halo 3 Quick Start Home

11/23/2022 • 2 minutes to read

More tutorials for Halo 3 mod creation can be found here:

Content Creation Quick Start

This article provides an overview of Halo 3 modding tools and the structure of folders used during modding.

Designer Boot Camp Section 1

This part of the Designer Boot Camp teaches you how to gather weapons, characters, and objectives to create a playable scenario.

Designer Boot Camp Section 2

This part of the Designer Boot Camp teaches you how to set up objectives and scripts to create a playable scenario.

Quick Start Overview

11/23/2022 • 13 minutes to read

This article provides an overview of Halo 3 modding tools and the structure of folders used during modding. In addition, it contains the diagram of the general modding pipeline for the creation of a level. For the full description of this pipeline, please refer to the Creation of Simple Level doc.

Halo 3 Modding Tools

This section contains brief descriptions of the modding tools that are included in the **Halo 3 Editing Kit (H3EK)**.

In the root folder of this package, you will find executables of all these tools, except the 3D modelling package (such as Blender, 3DSMax, or Maya) that needs to be installed separately.

Once downloaded, Halo 3 modding tools do not require any extra installation steps. You can simply launch them from the appropriate folder.

Tool and Tool_Fast: "Aggregators" for Commands

The **tool** and **tool_fast** apps (**tool.exe** and **tool_fast.exe**) are command-line programs that are very important in the world of Halo 3 modding. Roughly speaking, we can say that these two apps are slightly different implementations of the same concept – the necessity to have some central point for launching various commands. Both of these apps aggregate a large set of sub-commands that are necessary for the different parts of the modding pipeline.

The most important ability of **tool** (tool.exe) is to transform source data (level geometry and models in the format of **FBX** files, images, text, etc.) to the format used by the engine. For example, you can generate an asset file (**ASS**) with the static geometry of the level from an FBX file using the **fbx-to-ass** command of tool in the following format:

```
tool fbx-to-ass <fbx_file> <ass_file>
```

> **NOTE**
> However, the tool app is used not only for that. You can list all of the sub-commands available in tool by launching tool.exe in the command line prompt without parameters.

The **tool_fast** app (**tool.exe**) is an optimized version of **tool** that supports multi-threading, which allows it to perform the same operations faster. Some commands of **tool_fast** duplicate the commands of **tool**, and we recommend you to use **tool_fast** commands in this case. Commands inside **tool_fast** should be sufficient for most popular modding tasks. However, tool_fast displays fewer errors and warnings, so if you need to do debugging or have some issues, switch back to **tool**.

Sets of sub-commands within **tool** and **tool_fast** are slightly different and some commands exist only in one tool or the other. For example, the **structure** sub-command exists only in the tool_fast. This command allows you to build a **.scenario** tag file of the level for Sapien from the **ASS** file. The format of this command is the following:

```
tool_fast structure <ass-file>
```

> **NOTE**
>
> Similar to the tool app, you can list all sub-commands of tool_fast by launching tool_fast.exe in the command line prompt without parameters.

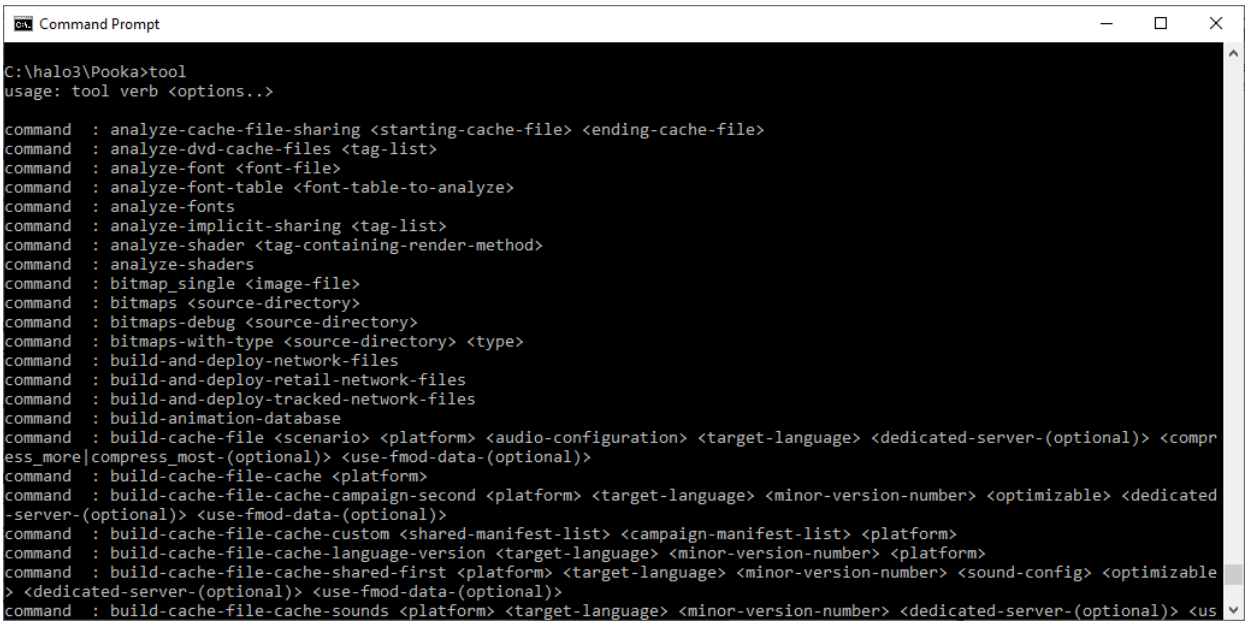


Fig 1. Sub-commands of tool (part of the full list).

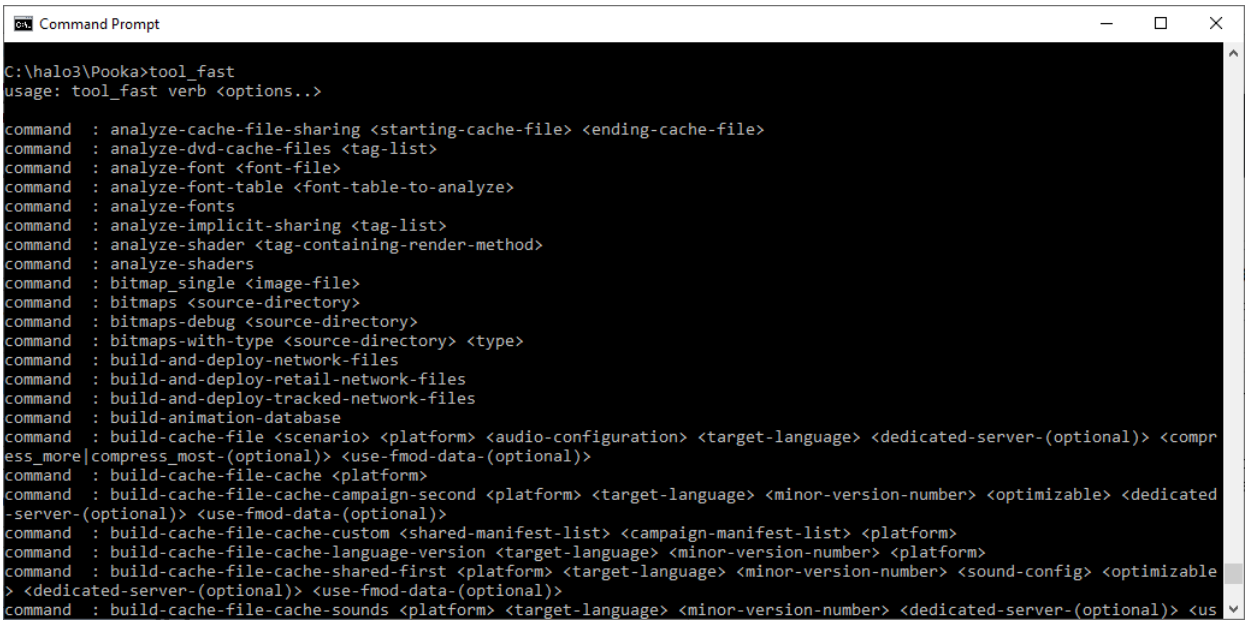


Fig 2. Sub-commands of tool_fast (part of the full list).

Guerilla and the concept of "Tags"

Guerilla (guerilla.exe) is used to edit the "tags" of modding entities. The concept of "tags" is a cornerstone of Halo 3 modding.

A single Tag can be described as a set of properties that describe the game entity of a particular kind to the game engine. Please note that these properties can link to some resources (textures, files, etc.) called tag's resource references. There are tags for levels (scenarios), game objects, AI, shaders, sounds, textures, network properties, effects, global variables, UI, cinematics, and so on. Almost every source object of the game is transformed into a single or multiple tags in the modding pipeline. However, not all game properties are editable as tags, and not all properties of individual tags are editable. This is done intentionally since modification of some aspects of the game can lead to incorrect behavior and crashes.

Every data tag in the game can be opened and viewed in Guerilla.

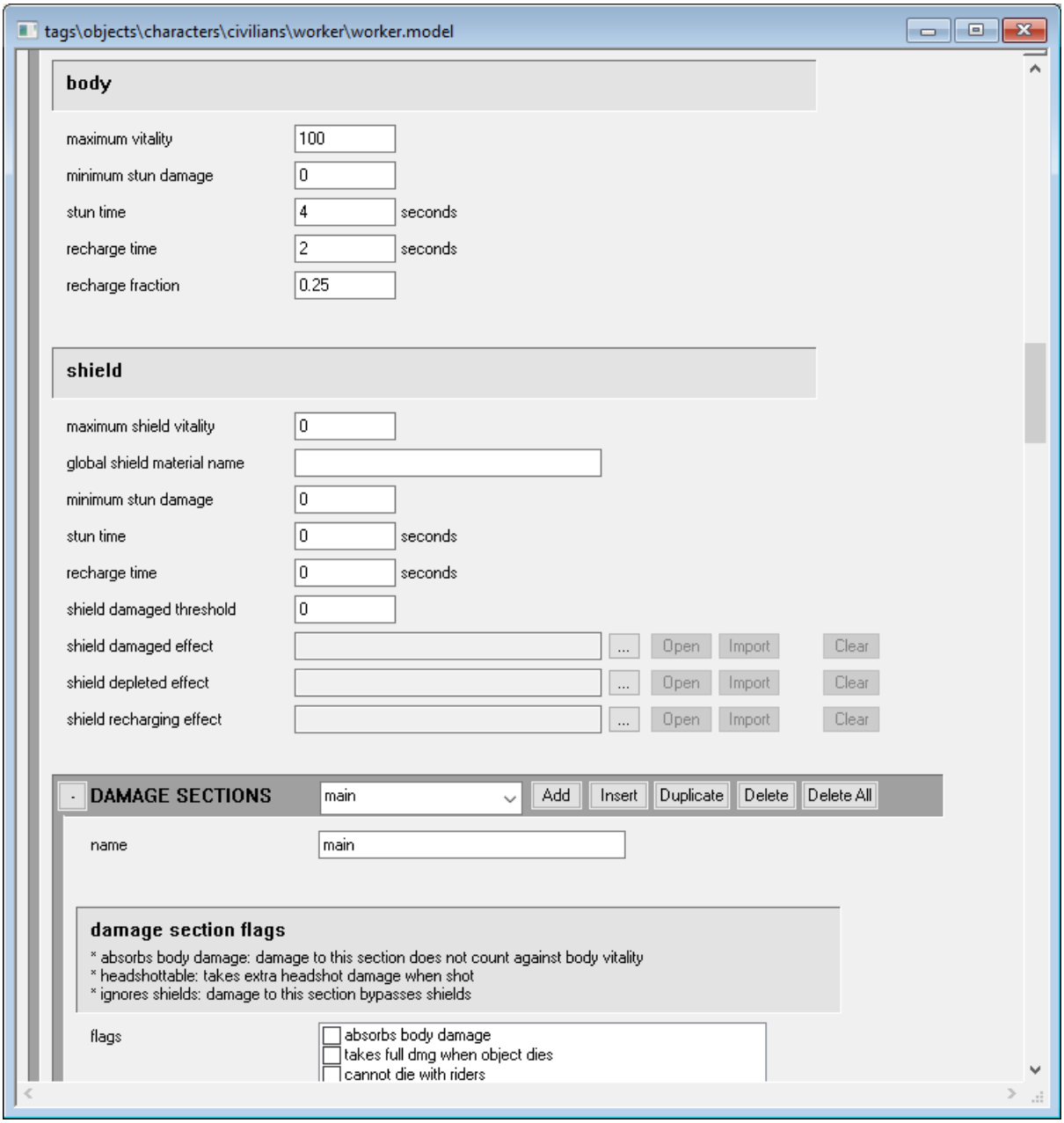


Fig 3. A window in the UI of Guerilla with a particular tag opened.

Sapien: Visual Level Editing

Sapien (sapien.exe) is a visual editor for Halo 3 levels (they are called "scenarios" in Halo 3 modding). It allows you to fill your scenario with objects, set up lighting, spawn points, AI used on the level, etc.

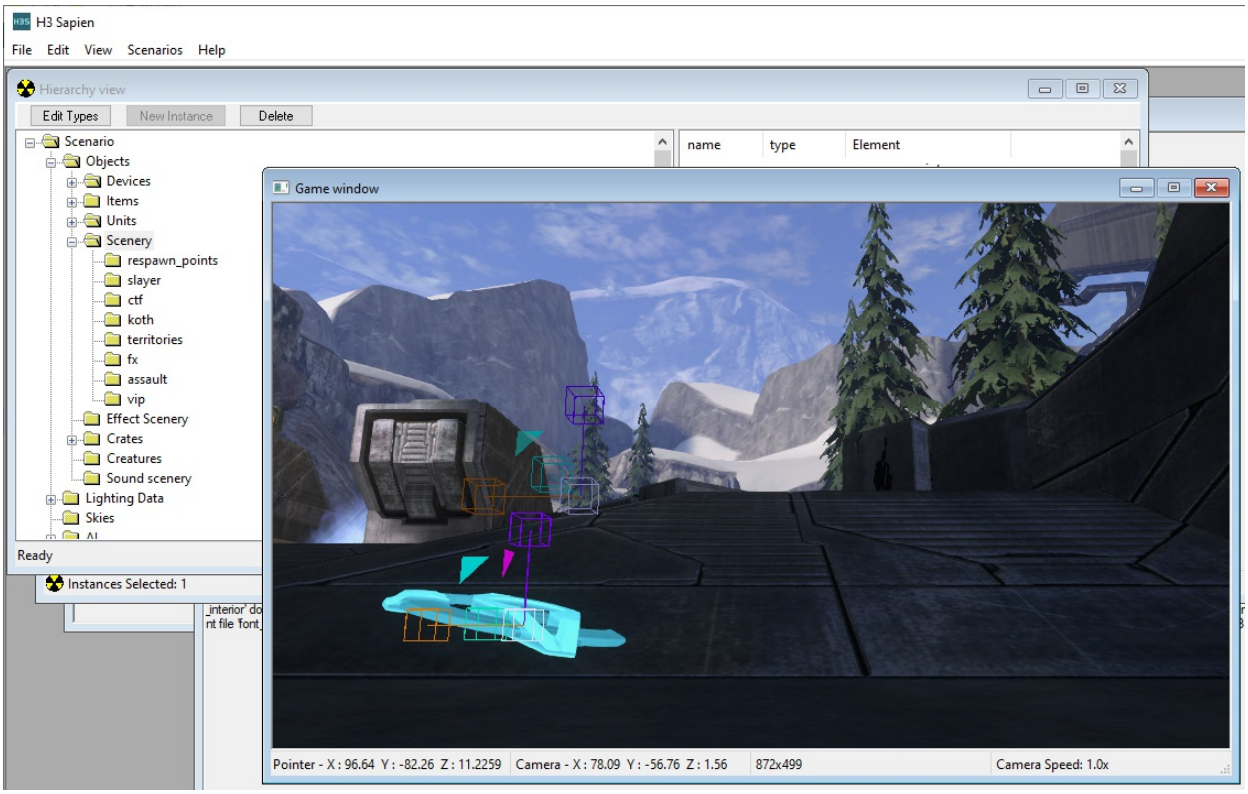


Fig 4. Editing a level in Sapien, part of the UI.

> **NOTE**
>
> Modding tools for some previous versions of Halo used photon mapping (direct simulation of real-life light particles) to calculate lighting of the level. It was done in **tool** by the specific **lightmaps** sub-command, currently obsolete. Now, lighting is calculated using the **calc_lm_farm_local.py** python script, see below.

halo3_tag_test.exe: Game Version Loaded from Tags

halo3_tag_test.exe is a specifically modified version of the Halo 3 game that is able to load itself not from regular game files, but from tags (see 2.2 above for a definition of tags).

Please note that this version of the Halo 3 game can load itself only this way.

The main purpose of this application is local testing of the tags you have created from the source files, tags' resource references, and so on, without doing a final build of the level/model.

To be able to launch **halo3_tag_test.exe** you will need to create its configuration file – **init.txt**, in the same directory. The app will execute Halo Script commands specified in this file and load settings from it. So, in this file, you will need to specify such parameters as the starting level, necessary debug parameters and cheats, and so on.

In its most basic setup, the **init.txt** file must contain only the staring level as a parameter, which is specified by the **game_start <path_to_level_from_the_"tags"_folder>** command.

For example:

```
game_start levels\multi\riverworld\riverworld
```

The full list of commands that can be used in this file can be obtained by opening the Halo 3 console in the game (type "~" to open it) and executing the **script_doc** command in this console. This will generate the **hs_doc.txt** file with the command descriptions in the folder of **halo3_tag_test.exe**.

Along with commands, in this file, you can set values of various game settings, such as cheats and debug

options. For example, **cheat_deathless_player on** – will enable invulnerability, **cheat_infinite_ammo on** – will give infinite ammo, **debug_objects on** – will enable the debug visualization of the game objects, models, and collision models.

> **NOTE**
> For details on this, see Console Commands.

When launching **halo3_tag_test.exe** you may want to change the resolution of the game. It can be changed by the **-width** and **-height** parameters of the executable that can be specified in the command prompt. You can omit one of these parameters (the default 16x9 ratio will be used in this case).

For example

```
halo3_tag_test.exe -width 1280 -height 720
```

During its launch, **halo3_tag_test.exe** will display errors, warnings, and other debug info.

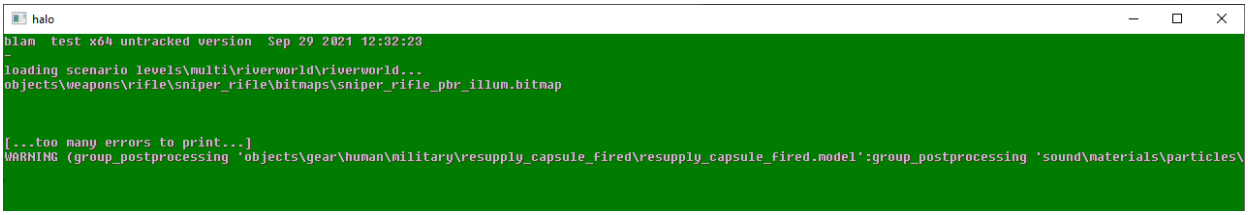


Fig 5. halo3_tag_test.exe loading a level.



Fig 6. Loaded level in halo3_tag_test.exe (with some debug visualization enabled).

calc_lm_farm_local.py: Script that Calculates Lighting

The **calc_lm_farm_local.py** file is a Python script that is used for the calculation of lighting of the level.

This script simply executes appropriate commands from **tool_fast**, so you can calculate lighting without it, if necessary. However, the script simplifies this process for you.

As any other script in Python, **calc_lm_farm_local.py** requires Python to be installed for the execution. You can

install Python from https://wiki.python.org/moin/BeginnersGuide/Download.

> **NOTE**
> Versions greater than Python 3.6 should be used for launching this script.

After Python is installed, you will be able to launch the script and calculate lighting by the command in the following format:

```
python calc_lm_farm_local.py <scenario tag file, w/o .ext> <bsp name> <quality> (<light_group>)
```

For example:

```
python calc_lm_farm_local.py levels\multi\s3d_turf\s3d_turf s3d_turf high
```

> **NOTE**
> A level can be divided into multiple BSPs (Binary Space Partitions) during its creation in the 3D modelling program. However, most of the custom levels have a single BSP only. Large campaign levels may have multiple BSPs.

> **NOTE**
> <light_group> can be omitted since **all** light groups are calculated by default.

3D Modelling Program of Your Choice

Along with these tools, you will probably need the 3D modelling program for modding. The following popular 3D programs can be used for that:

- 3ds Max (https://www.autodesk.com/products/3ds-max/overview)
- Maya (https://www.autodesk.com/products/maya/overview)
- Blender (https://www.blender.org/)

All of these tools support FBX (.fbx) files as an export format. These FBX files can be used as the source files for the creation of levels and models.

> **NOTE**
> Support of the **FBX** format as a data source is a new part of the Halo 3 modding pipeline. Due to this feature, you can perform modelling not only in 3ds Max as before, but, also, in Maya, or even in an open-source and free 3D modelling tool – Blender.

Main Folders Used for Modding

In the root folder of the modding package for Halo 3, you will find executables of all modding tools mentioned above.

During modding with these tools you will use the following main folders:

- **data**
- **tags**

- **maps**

The **data** and **tags** folders are initially provided in the form of **.zip** archives, so you will need to extract them to create the **data** and **tags** folders used by the modding tools.

"data": Source Folder

The **data** folder is intended for source data.

In the modding package itself, this folder is almost empty since we do not provide all initial source files of game data, only their tags (see below). However, you will probably gradually fill this folder during your modding activities.

> **WARNING**
> Please do not delete the initial contents of the **data** folder. They are necessary for the game engine to be able to start the game and load all of the provided maps. Particularly, the initial contents of **data** include some game scripts and XML descriptions of levels.

During the process of modding, the **data** folder should contain the source files of the levels and models, various resource files (textures, sounds, etc.), and so on.

For contents of this folder, you must follow the same folder structure as exists in the **tags** folder (except **temp_scenario**):

- **ai**
- **camera**
- **cinematics**
- **effects**
- **fx**
- **globals**
- **levels** – this folder will contain source data for your levels. Please note that if you want to make some resources common for multiple levels, we recommend you to put them into the **shared** subfolder.
- **multiplayer**
- **objects** – this folder will contain source data for your objects (models, their animations, etc.)
- **rasterizer**
- **shaders**
- **sound**
- **ui**

> **NOTE**
> For descriptions of all folders in this folder structure, please refer to the description of the tags folder (see tags below).

File formats that are stored in subfolders of **data** include the following:

- Source files of modelling scenes (from 3ds Max, Maya, or Blender) – these are exported to FBX (see below) by the means of the used 3D modelling program.

- **FBX** files – act as source files for ASS, JMS, and JMA (see below).
- **ASS** files – static geometry of the level.
- **JMS** files – files of the model. Please note that each model has three JMS files (or two, if a model has no constraints). These are located in the objects folder, in the sub folder of the model in the objects hierarchy and these JMS files must be put into the separate subfolders (see below). For example, the path to the "render" JMS for Cortana will look like this: data\objects\characters\cortana\render\cortana.jms.
 - **render** – in this folder, you should put the JMS file with visible geometry and the skeleton of the model.
 - **collision** – in this folder, you should put the JMS file with collision objects of the model.
 - **physics** – in this folder, you should put the JMS file with constraints (i.e. physical joints) of the model. This folder/JMS is optional since not all models have constraints.
- **JMA** files – animation files.
- **TIFF/TIF** files – source files of textures/images.
- **WAV** files – source sound files.

> **NOTE**
> One source file from the data folder can correspond to multiple tag files in the **tags** folder. And, vice versa, multiple source files can be used to generate a single tag file.

"tags": Folder with All Tags

The **tags** folder is used for tag files of various types. Roughly speaking, tag files correspond to the particular game entities in the format of the Halo 3 engine. Please note that a single source file can correspond to multiple tag files generated on its basis. For example, the conversion of a JMS file (of a model) will result in multiple tag files.

> **WARNING**
> Please do not delete the initial contents of the **tags** folder. They are necessary for the game engine to be able to start the game and load all of the provided maps.

The folder structure within the **tags** folder is the following:

- **ai** – This folder contains tag files related to default behavior that is common for some types of AI encounters (both with friendly and enemy NPCs). Please note that the AI encounters themselves are described in the scenario tag files in **levels** (and these encounters should be edited in Sapien, not in Guerilla).
- **camera** – This folder contains tag files related to vehicle camera movement only.
- **cinematics** – This folder contains tag files related to cinematics.
- **effects** – This folder contains a few legacy tag files related to particle effects. Please use the **fx** folder for your effects, but do not delete the old legacy effects from the **effects** folder since they are still used by the game engine.
- **fx** – This folder contains tag files related to various special effects.
- **globals** – This folder contains tag files related to the global parameters of the game and all its subsystems.

- **levels** – This folder contains tag files related to levels. Please note that it contains the **shared** subfolder, which contains tags for resources that are shared between all levels of the game.
- **multiplayer** – This folder contains tag files related to the setup of the multiplayer in the game. Please note that this folder does not contain the multiplayer levels. Multiplayer levels are in **levels\multi**.
- **objects** – This folder contains tag files related to various objects: characters, vehicles, weapons, skies of the levels, objects' physics, and so on. Please note that objects related to particular levels are in **objects\levels**.
- **rasterizer** – This folder contains tag files related to global shaders that are *not dependent on materials*.
- **shaders** – This folder contains tag files related to shaders that are *dependent on the materials*.
- **sound** – This folder contains tag files related to various sounds.
- **temp_scenario** – a temporary folder of Sapien. If you have closed Sapien without saving your changes, it will restore them from the temporary files in this folder.
- **ui** – This folder contains tag files related to UI used by the game.

> **NOTE**
> Except for the minor changes described above, the folder structures of **tags** and **data** *must* be similar for your custom files. This is necessary for the correct operation of modding tools. Along with it, this will allow you to found your source files easily when you know their tag files and vice versa.

There are a lot of different types of tag files in the game. Most of them have extension names that speak for themselves. For example, **.scenario** tag files correspond to the main tag files of the level, **.bitmap** tag files correspond to textures and images, and so on.

"maps": Folder with Compiled Maps

At first, there is no **maps** folder in the root folder of the modding package for Halo 3. It will appear as soon as you build the cache files for your first level.

During your work with the level, the **maps** folder may contain the following:

- **.map** files – the cache files for the scenarios. These files contain all contents of the level.

The **.map** files are generated by the **build-cache-file** command of **tool_fast** based on **.scenario** tag files.

To display possible parameters for this command, you can execute the **tool_fast build-cache-file** command in the command prompt. In its simplest form the command requires only the name of the **.scenario** file (without the extension) and the type of the target platform (e.g. **x64**):

```
tool_fast build-cache-file <scenario> <platform>
```

For example:

```
tool_fast build-cache-file levels\multi\riverworld\riverworld x64
```

After the execution of this command, the tool app will start to build the .map file.

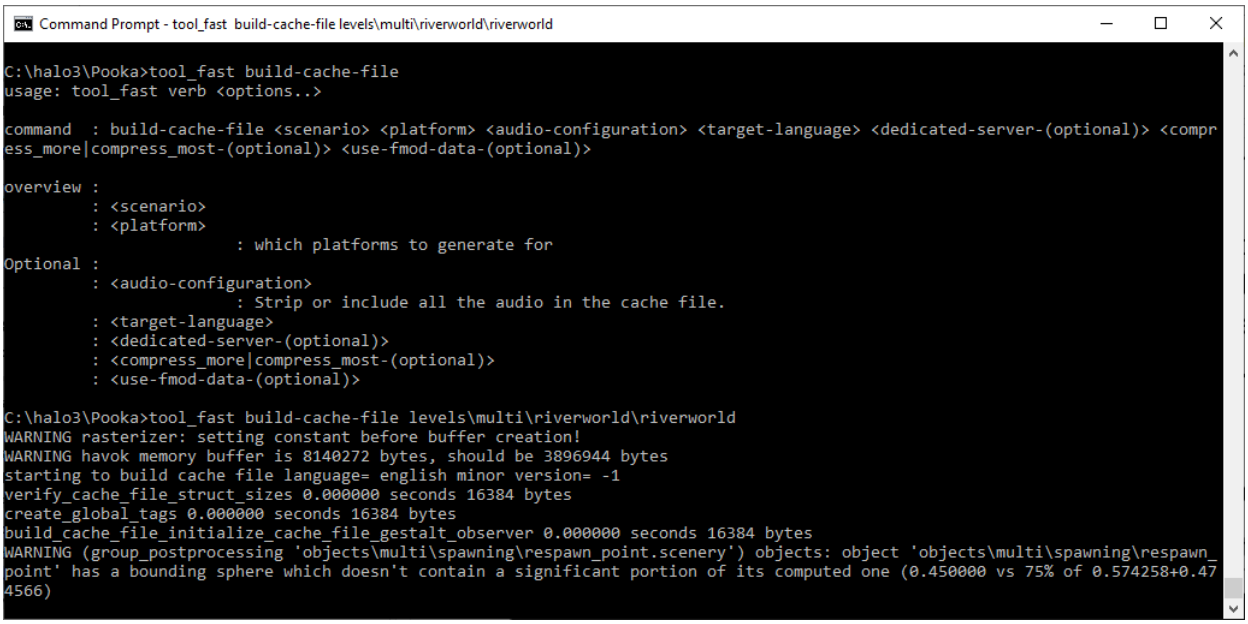


Fig 7. Building .map file.

Level modding Pipeline: Basic Diagram

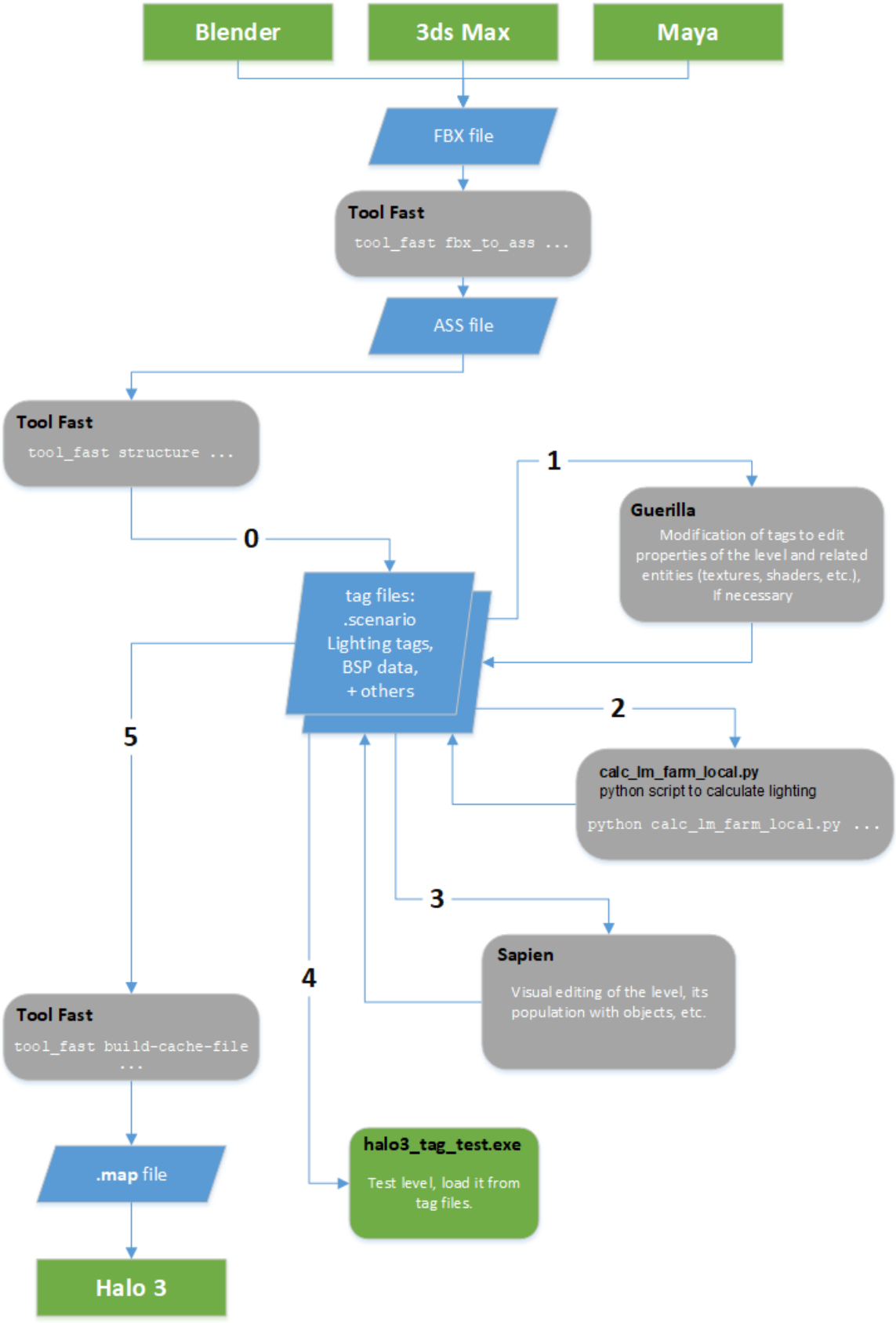


Fig 8. General pipeline for the creation of a level.

For full details of this workflow, please refer to the creation process.

Quick Start Process Step 0 - Prerequisites

11/23/2022 • 2 minutes to read

To pass all steps of this tutorial you will need the following:

- **Halo 3 Editing Kit (H3EK)**
- **Blender**

Halo 3 Editing Kit is available at Steam, under the "**Halo 3 Mod Tools - MCC**" title. As other games from Steam, it will be installed to the "**steamapps**" folder, i.e. its default installation path is typically something similar to "**C:\Program Files (x86)\Steam\steamapps\common\H3EK**". The precise path may vary, but you can always open its installation directory from Steam, using the Local Files feature: right-click the installed game in **LIBRARY**, select **Properties**, in the appearing window switch to **LOCAL FILES** tab, and then click **Browse**.

> **NOTE**
> Most commands of this guide need to be launched from the folder which H3EK is installed to, e.g. from the "**C:\Program Files (x86)\Steam\steamapps\common\H3EK**" by default. So, it might be a good idea to make a shortcut for this folder.

You can download and install Blender from https://www.blender.org.

> **NOTE**
> You can use such tools as **3ds Max** and **Maya** to accomplish Step #1 and Step #2 of this tutorial and then follow other steps without changes. Please note that export settings in these tools Step #2 will be different.

If you have both H3EK and Blender, you can proceed to Step #1: Modelling Level Geometry in 3D Tool.

Quick Start Process Step 1 - Modelling Level Geometry in 3D Tool

11/23/2022 • 23 minutes to read

As a sample level, we will create a simple hollow box (room) with a sky above and a single source of light.

> **NOTE**
>
> The description below corresponds to a freshly installed Blender 2.93 with a default layout of panels. If you have intentionally modified the default layout, then you probably know where to find the necessary panels or, if you occasionally modified the layout, you can reset it: https://blender.stackexchange.com/questions/48687/how-do-i-restore-the-original-window-configuration.

To create this level, do the following:

1. Launch Blender.
2. If necessary, in Blender, select **General** in the **New File** section of the startup dialog.

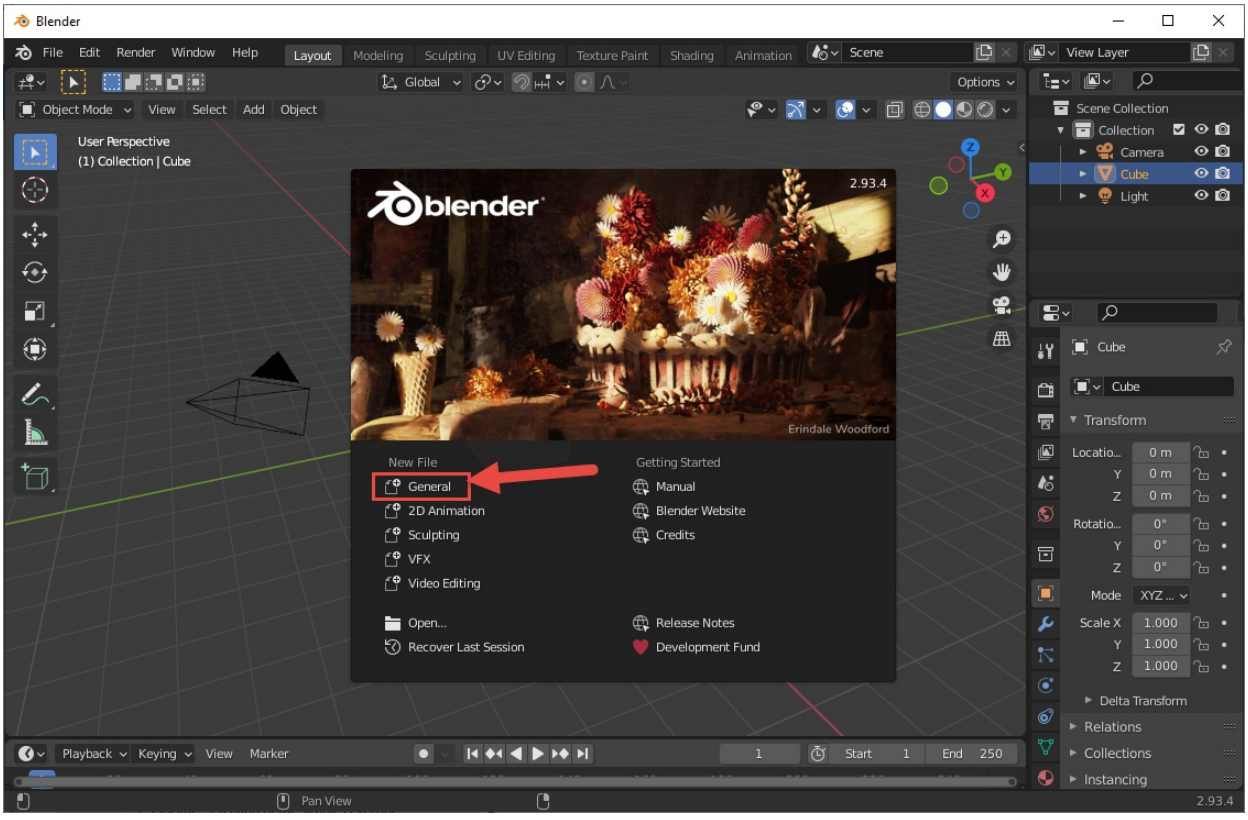


Fig 1. View of the the initial Blender start screen and the General New File option.

3. In the **Outliner** in the upper right corner of the screen, select all default contents of the scene (**Collection, Camera, Cube, Light nodes**) and delete them by pressing DEL. Now, you have an empty scene in your Blender.
4. First of all, we need to create a ***frame node***. This node will be the origin of your level and it will contain all objects that will belong to the level. This node is necessary for the creation of the FBX file that will be appropriate for the game engine. We will create a frame node in the form of the sphere (any shape is actually valid for that, but a sphere is easier to distinguish from the other shapes that will eventually be in

the level).

To create a frame node:

- In the left upper corner of the **3D Viewport**, click on the **Add menu** and select **Add > Mesh > UV Sphere** to add a sphere to your scene.

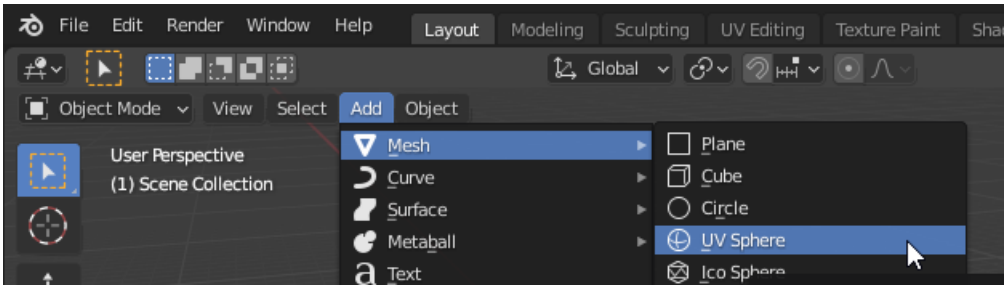


Fig 2. View of the UV Sphere option in the Add menu.

- Double-click the node of this sphere and change its name to **b_levelroot**.

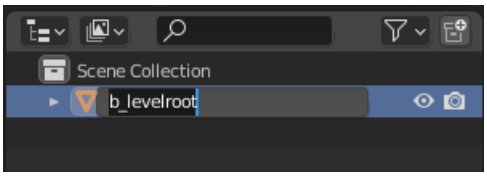


Fig 3. The UV Sphere with its node name changed to b_levelroot.

> **WARNING**
> It is important that you follow this naming convention, since these names of nodes will be the same in the FBX file and, then, in the ASS file, and, finally, the game engine will use them to identify the types of nodes and their properties. If you do not name the frame node correctly, the environment will not be oriented correctly in the game. Object and material names are important to Halo 3 modding, so get used to paying attention to this and understanding what the different keys are.

> **NOTE**
> The **b_levelroot** name is the most typical name for the frame node. However, only the **b_** prefix from it is mandatory for the identification of the frame node in the Halo 3 tools and engine. The list of all such prefixes is the following:
>
> - **b_*** (e.g. **b_node**)
> - ****b **** (e.g. ***b node**)
> - **bone_*** (e.g. **bone_node**)
> - ****bone **** (e.g. ***bone node**)
> - **frame*** (e.g. **frame node**)
> - **bip01*** (e.g. **bip01node**)

- Ensure that your frame node is located at the origin of the scene. If it is not so, change its **Location** to **(0, 0, 0)** in the **Transform** panel at the right side of the Blender window.

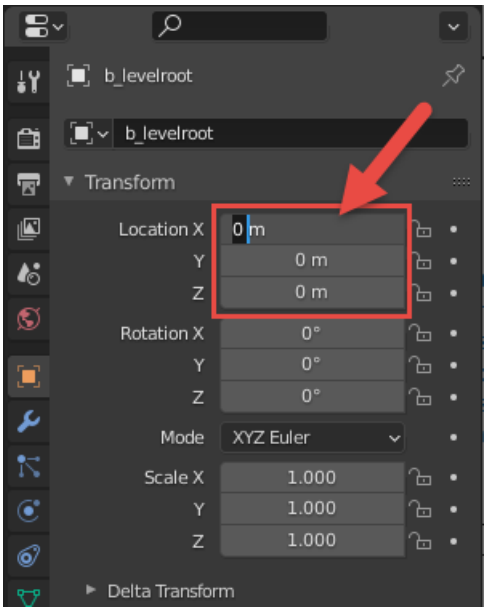


Fig 4. Location transforms set to (0, 0, 0).

> **NOTE**
>
> Most levels have their frame node at **(0,0,0)** for simplicity. But you can also place it somewhere off to the side, out of the way, if necessary. However, it is very important not to move it later on, or you might discover all your player and weapon spawns moving along with it.

5. Now, you can start to create a level environment itself. Particularly, you can create a box (cube) those faces will be the floor, walls, and the sky. The box will be hollow – i.e., the player will be placed inside it and will see the internal faces of it (not the external ones).

 To create such a cube:

 - In the left upper corner of the **3D Viewport**, click on the **Add menu** and select **Add > Mesh > Cube** to add a cube to your scene.

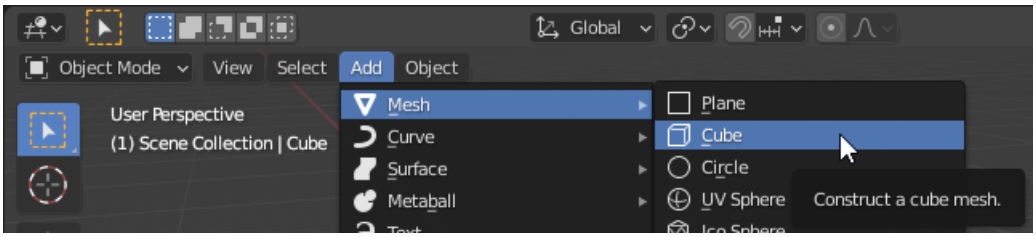


Fig 5. View of the Cube option in the Add menu.

 - For simplicity, let's keep the created box centered on all axes first. If your cube is not located in the origin, change its location to **(0, 0, 0)** in the **Transform** panel at the right side of the Blender window, as you did for the frame node above.
 - Now you need to scale your cube to make dimensions of your box appropriate for game characters to move within.

 If you switch to the 2-axes (orthographic) view by clicking one of the axes displayed at the **Navigation Gizmo** (located at the top right of the window, see the official help; you can switch back by holding the middle button of the mouse and moving the mouse to move the view), you will see that the default cube appears on the grid as a centered object of the **2 x 2 x 2** default units in size.

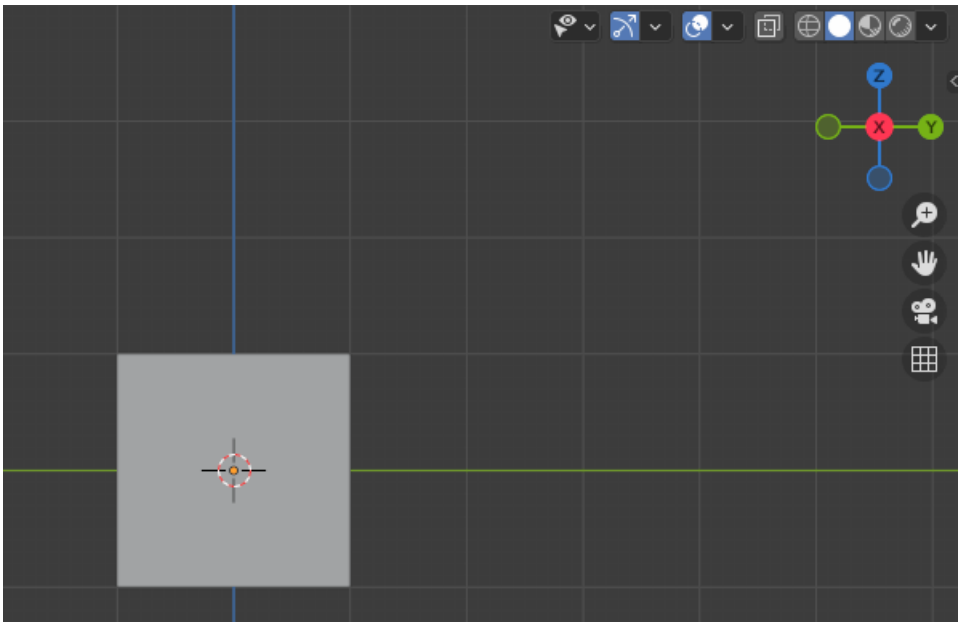


Fig 6. Orthographic view in Blender.

If metric units are selected, 1 blender unit (wide/height of the large cell in the picture above) corresponds to 1 meter. In Halo 3, the height of Cortana in these units is approximately 1.5 (i.e. she is ~1.5 meters high).

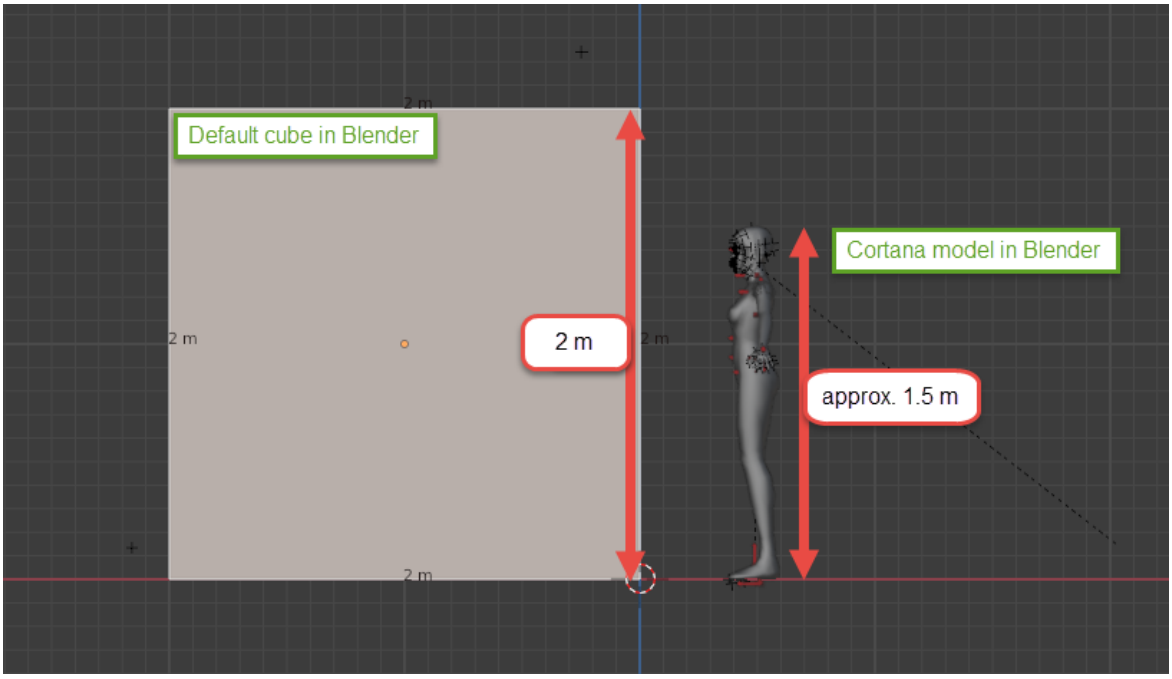


Fig 7. Cortana model height compared to the newly created cube.

> **NOTE**
> Yes, we do agree that Cortana is, in fact, a projection and has no height at all, but her model is her model.

And, the height of Master Chief is approximately 2.1 (i.e. he is ~2.1 meters high) in these units.

So, let's scale the level to make it appropriate for at least minimal moving of characters. For example, let's make it **40** times larger, which will make it **40*2 x 40*2 x 40*2 = 80 x 80 x 80** default units in size (i.e. the overall width/height/depth of the cube will be 80 meters in Blender).

NOTE

Regardless of the units and unit system you select in Blender for modeling, you should take special care of units selected during export to FBX (see the next section for details). Particularly, in this tutorial, we are using the Metric unit system and Meters for length while modeling and we will convert them to Halo 3 units at Step #2.

- To scale your cube:
 - If the cube is not selected, select it (by clicking).
 - Change its **Scale** to **(40, 40, 40)** in the **Transform** panel at the right side of the Blender window. (After that you will probably need to zoom out using the mouse wheel.)

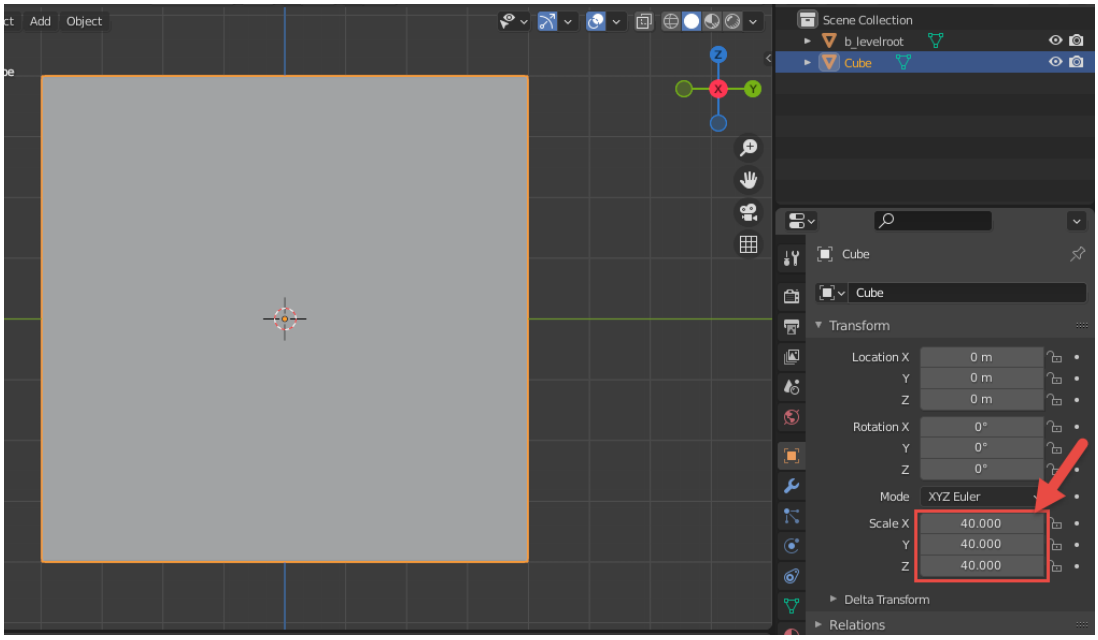


Fig 8. Cube scale is set to 40 on each axis.

- And, it's a good idea to move the "floor" of your box to the 0 too, for convenience. The mapping of Blender axes to the directions in the game is the following: Z-axis in Blender corresponds to the Up direction in Halo 3, X-axis – corresponds to the Forward direction, and Y-axis – to the Leftwards direction. So, let's move the floor of your cube to 0 by Z-axis:
 - Click the Y-axis in the **Navigation Gizmo**. Blender will show you the ZX-view, with the Z-axis pointing upwards. Keep the cube selected, if it is not selected – click it to select.

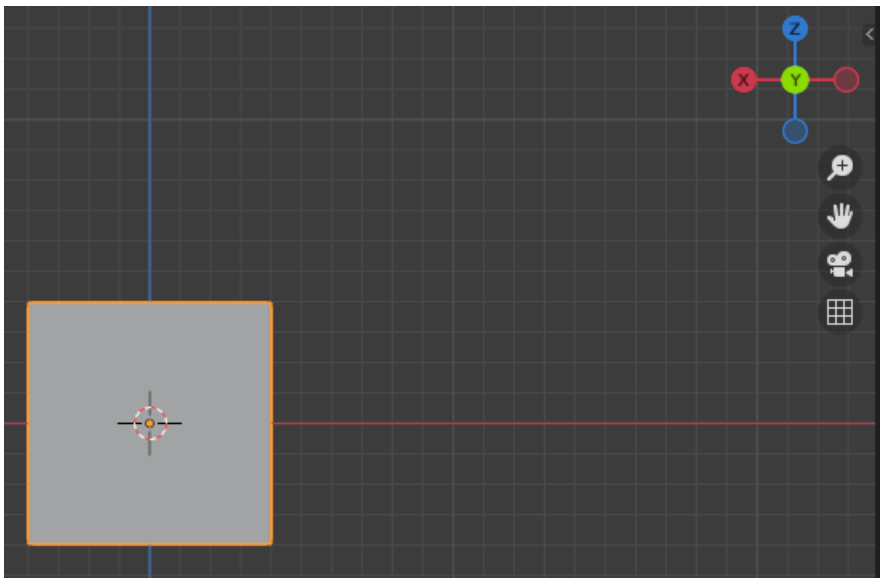


Fig 9. Orthographic view in Blender on the Y axis.

- In the Toolbar (on the left side of the editor area), select the **Move** tool.

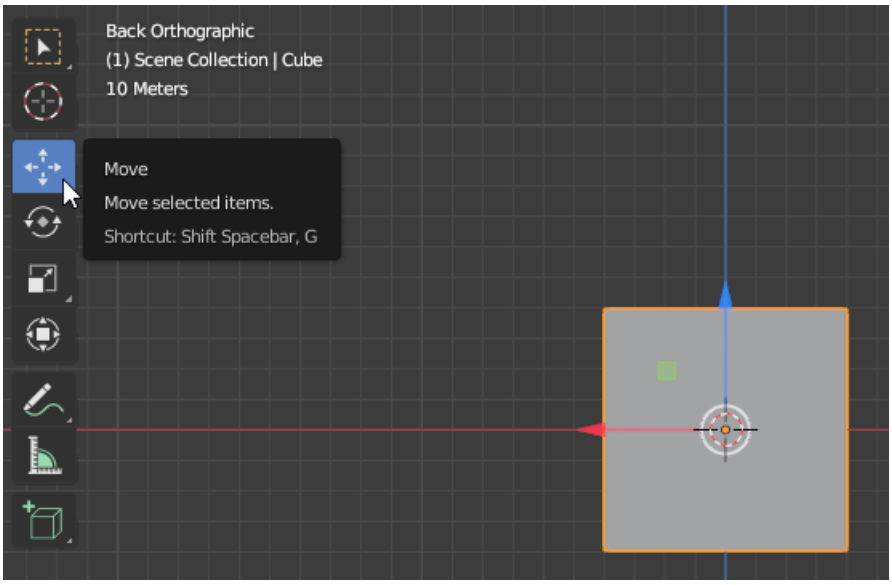


Fig 10. The Move Tool in the Blender toolbar.

- By dragging the blue arrow (corresponding to the Z-axis pointer of the cube) move the cube upwards for 40 blender units.

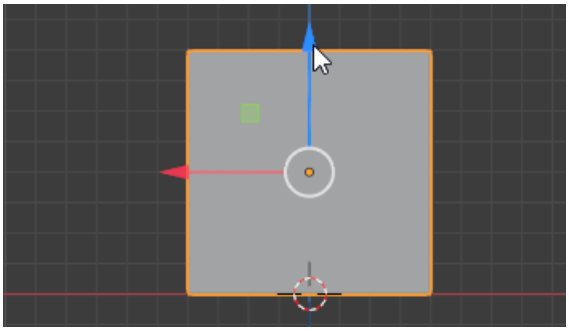


Fig 11. Moving the cube up 40 units on the y-axis.

If necessary, specify the exact location of the cube in the **Location** section of in the **Transform** panel to **(0, 0, 40)**.

- If you have not switched back still from the 2-axes view, you can do it by holding the middle button of the mouse and moving the mouse to move the view.

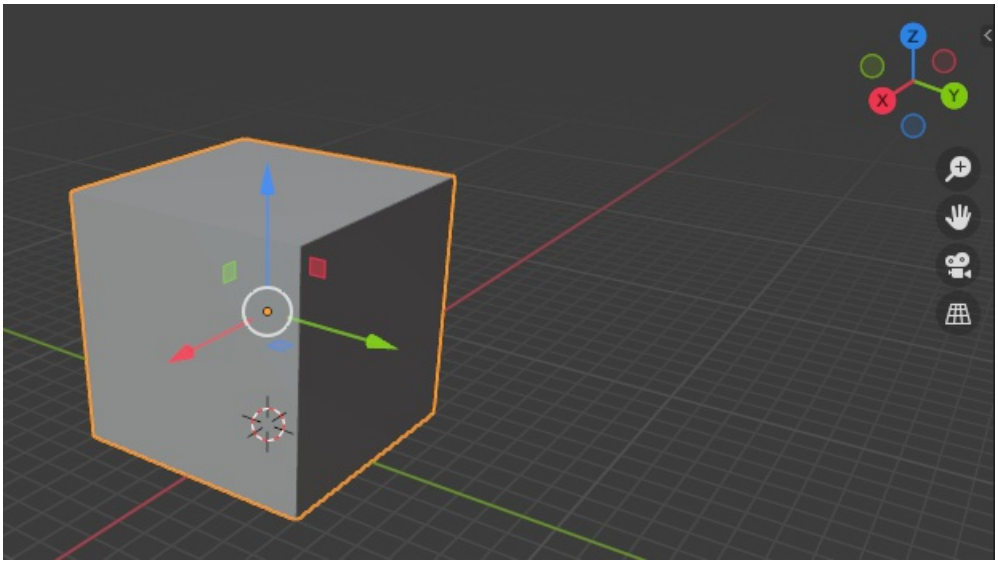


Fig 12. Two Axis view.

6. Now that both of our objects (frame node and the box) are positioned correctly, we need to link them together. Since all scene objects should be below the frame root node in the hierarchy, we need to place the cube under the frame node (i.e. to specify frame node as the parent bone for the cube).

 To do so:

 - In the **Outliner** (at the top right of the screen), select the node of the **Cube**.
 - Then, while holding CTRL to enable multi-select, select the future parent node – the frame node (**b_levelroot**). It is very important to select the future parent object *last*.

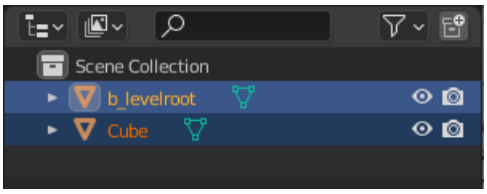


Fig 13. Object hierarchy with the cube and b_levelroot selected.

 - In the **3D Viewport**, right-click the selected objects and select **Parent > Object**.

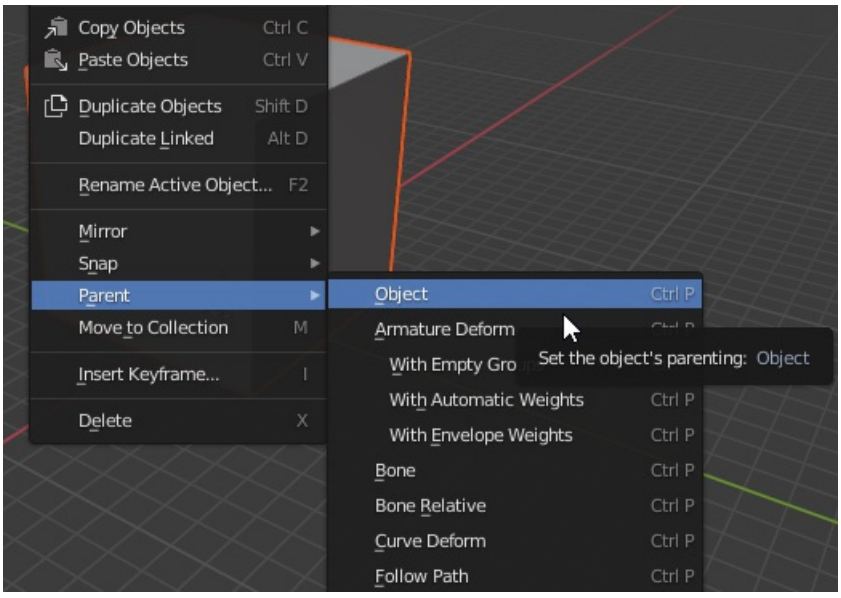


Fig 14. Parent > Object option within the right-click menu.

 - In the **Outliner**, after you expand the frame node (**b_levelroot**), you will see that the Cube is now under it in the hierarchy.

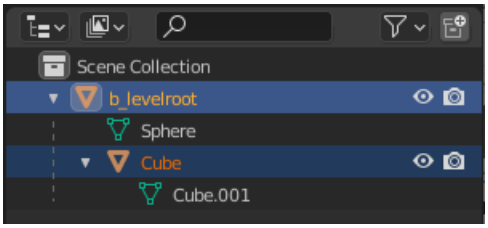


Fig 15. Expanded hierarchy with the Cube now parented to the b_levelroot.

7. Until now, we have done all the work to our environment on its outside. When we launch the map in the game engine, we want to be inside the box (cube). However, there is one important aspect here for the level to be rendered correctly – Normals. Normals tell Blender (and our game engine) which direction a piece of data is "facing". If you look at that data from the opposite side of the Normals, the data will not be rendered (visible) on the screen. I.e. the direction of the Normals sets the direction the object will be visible from. By default, for your cube, Normals are perpendicular to the face of each face of the cube and look in the outwards direction of every face. That's the opposite direction, since we need to be inside the

cube and need Normals to look inside of the cube.

So, let's flip the direction of Normals.

To do so:

- In the **Outliner**, select the **Cube** node.
- In the **3D Viewport**, switch from the **Object Mode** to the **Edit Mode** (at the drop-down at the top left of the **3D Viewport**).

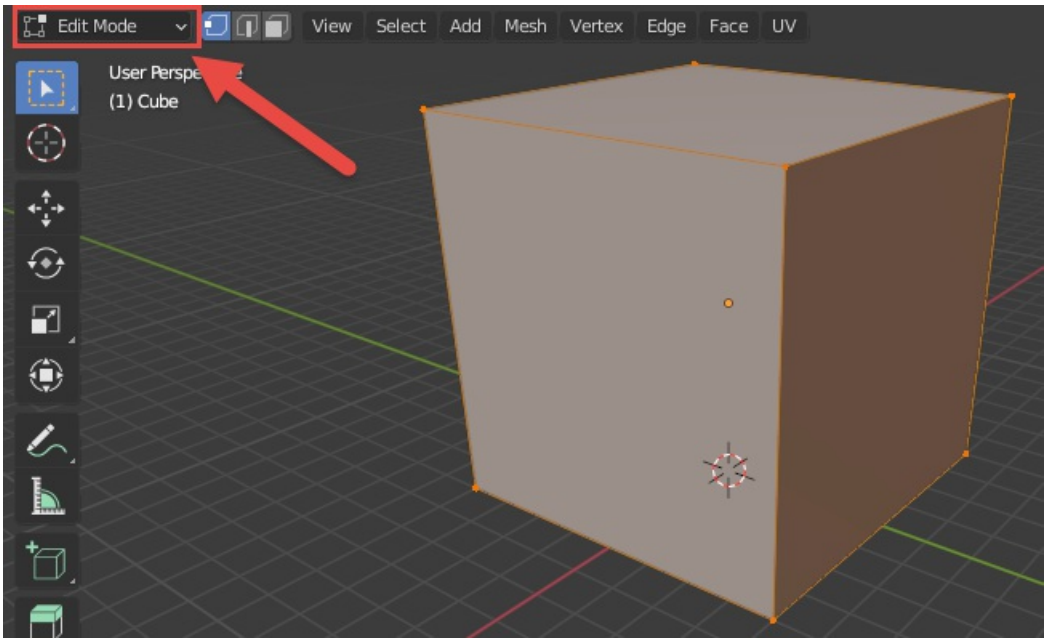


Fig 16. Edit Mode option in the UI.

- Click the **Mesh** menu (at the top of the **3D Viewport**) and select **Mesh > Normals > Flip**. Alternatively, you can press CTRL + N to access the **Normals** context menu and select **Flip** in it.

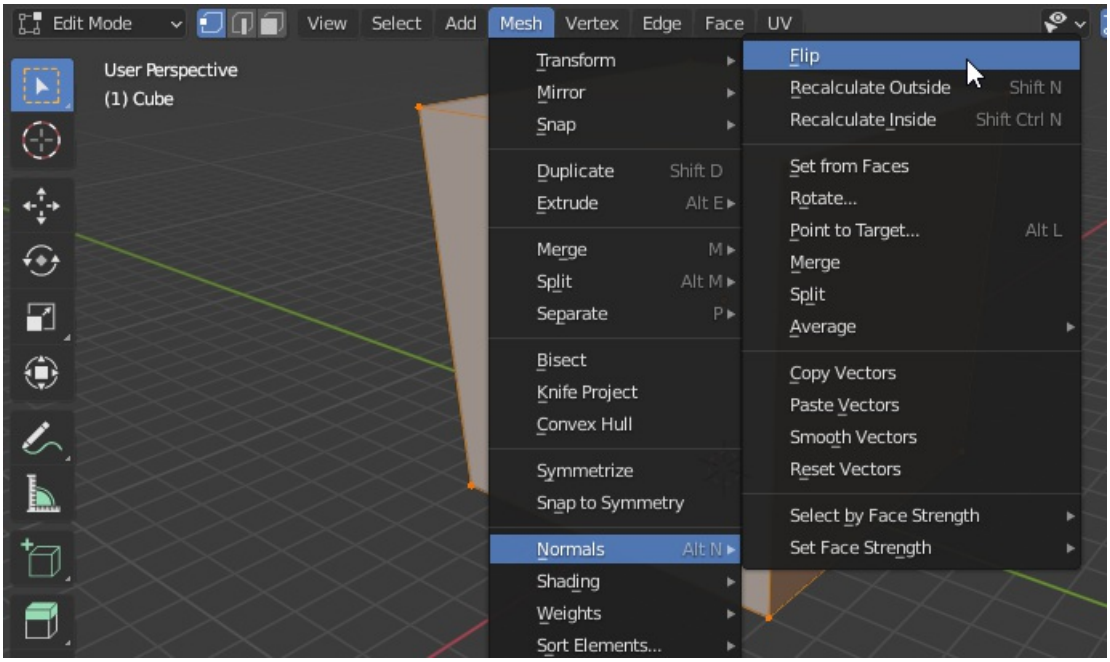


Fig 17. Normals > Flip option in the Mesh menu.

- To ensure that you have done it correctly, you can enable the **Backface Culling** option to distinguish the invisible back sides of the faces from their visible front sides.

 To enable this option:

- Open the **Viewport shading** menu (by clicking the drop-down arrow at the top right corner of the **3D Viewport**, see the screenshot below).
- Enable the **Backface Culling** option.

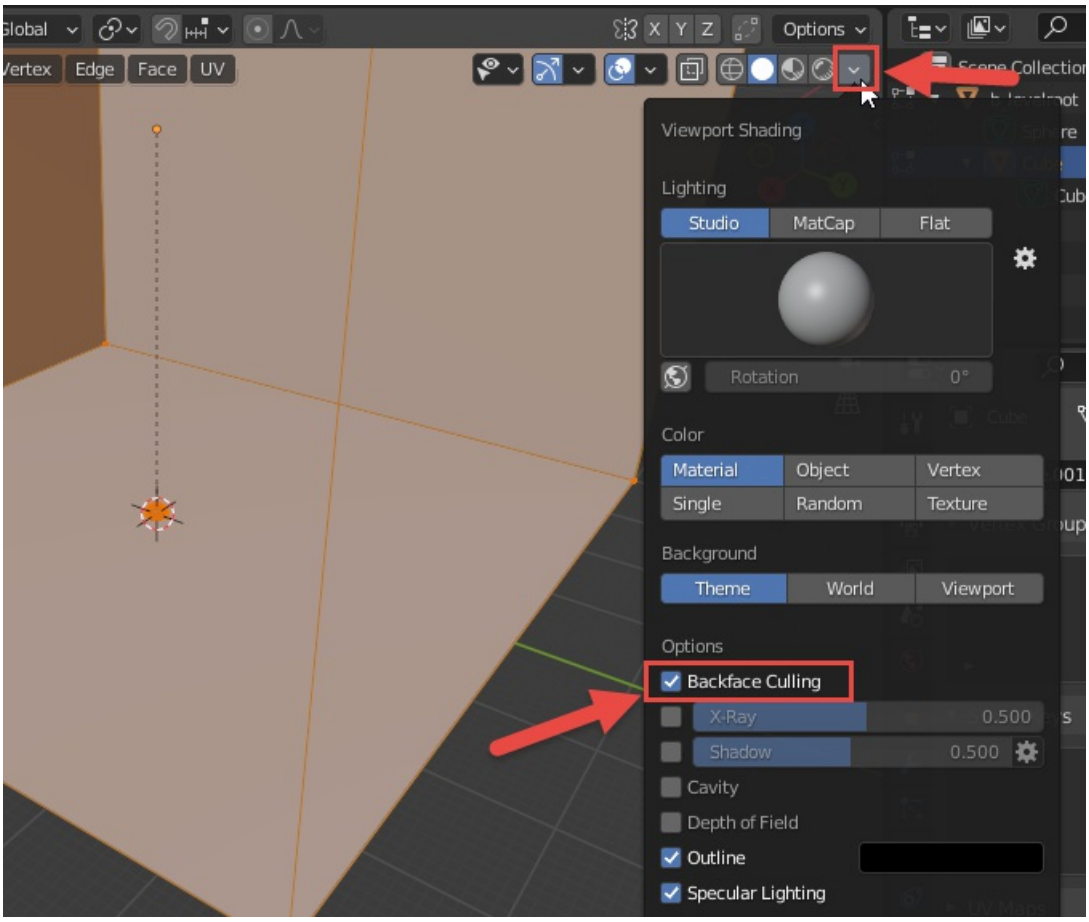


Fig 18. Backface Culling option in the Viewport Shading.

- After doing this, if you inverted normals correctly, only the inside sides of cube faces should be rendered with color in the viewport.

8. Now your cube will be looking like an empty "room" with walls. If you want to practice, you can add another small cube to the interior of this room, be repeating the previously described operations. However, for this small cube, the Normals step should be omitted, since you will be looking at this small cube from the outside, so the default direction of normals will do just fine. Don't forget to place this small cube under the frame node too.

 To add a small cube:

 - Switch back to **Object Mode**.
 - **Add > Mesh > Cube**.
 - If necessary, move this small cube somewhere inside the large cube (e.g. by **Move** tool or by setting **Location** in the **Transform** panel).
 - If necessary, scale this small cube (e.g. by changing **Scale** in the **Transform** panel).
 - Parent this small cube to the frame node (select the small cube; while keeping it selected, select frame node; right-click any point of the viewport and select **Parent > Object**).

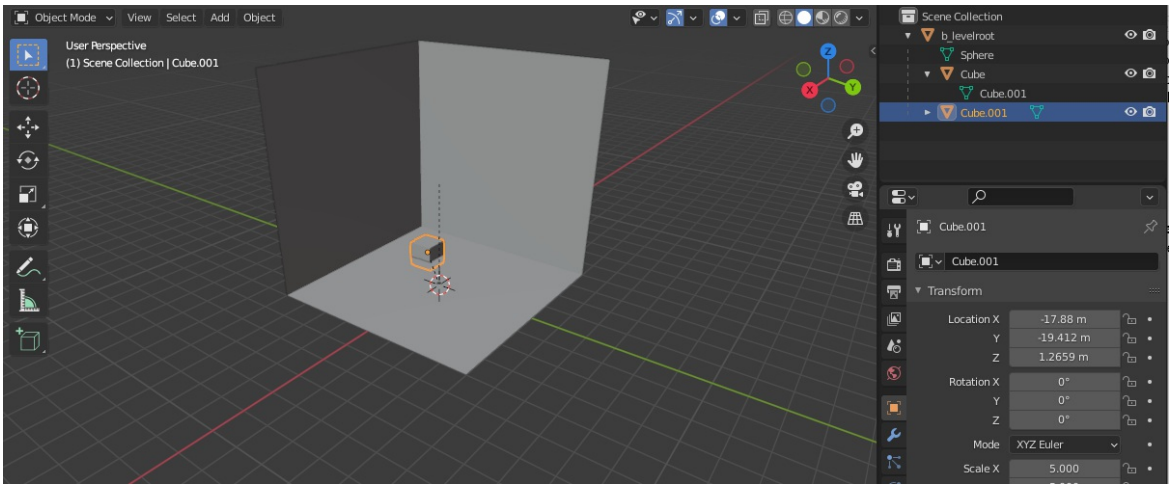


Fig 19. New cube in the center of the room.

> **NOTE**
>
> In this tutorial, we have changed the **Scale** of the small cube to **5** (i.e. it is 10m x 10m x 10m) and put its bottom slightly below the "floor" of the large cube to make it look a little bit prettier.

9. Now it is time to assign materials to these objects. Materials assigned to objects in Blender will define the properties of a surface in Halo 3. This will be done with the help of shaders that will be assigned to these materials later on.

> **NOTE**
>
> Note #1: In Halo 3, shaders are created and modified not in Blender, but in Guerilla, in the form of shader tags files. (However, assignment of textures for some specific materials, like **+sky**, is performed differently, not via shader tags, see Step #5 below for details). The properties of material defined by this shader tag will not only include the basic texture image to render on the surface but include any special rendering effects as well as other data such as the type of material it is in regards to physics and projectile impact particle effects. Some materials use individual shader tags that are created for them (with names exactly the same as the name of the source material), other materials use shared shader tags that are used for multiple materials. In general, shaders and shader tags are a rather complex topic and are out of the scope of this guide. However, since you will still need to assign a texture to one of the Blender materials of your level, you will create a very simple shader tag at Step #5 below.

> **NOTE**
>
> Note #2: In this tutorial, to keep things as simple as possible, we will not modify the UV maps of created Blender objects that will help to wrap these objects with your textures correctly. However, when you will be creating a highly detailed and professional level, you will probably need to work with these UV maps. As a very brief starting info on this topic, you can use Blender's official help: https://docs.blender.org/manual/en/latest/editors/uv/introduction.html.

In Blender, Materials are assigned to objects via *material slots* (see official help). I.e., at first, you create a material, then you select it in one of the material slots of the object, and this links this material to this object. If you want to assign a material to the whole object, you will need only one material slot with this material selected (in fact, if you are creating material from the object properties, it will become selected in this material slot automatically). However, if you want your object to use multiple materials (e.g. you want to assign one material to one part of the object and another material to another part), you will need multiple material slots, one slot per material.In our case, we will need two materials for our large cube: one for the "**+sky**" of it and one for the "**ground**" of it.

So, let's create and assign them:

- First of all, let's create and set up a **+sky** material. This is a special material that is applied to surfaces to render the skybox or sky model. Please note that the name of this material must be "**+sky**" (with a plus), it is a reserved name, and using this name the game will identify that this is the sky surface.

 We will use the **+sky** material for the "ceiling" and "walls" of the large cube.

 To create and set up this material, do the following:

 - Create the material:

 - Select the large **Cube** node and switch to the **Material Properties** tab in its **Properties** panel (in the lower-left corner of the window).

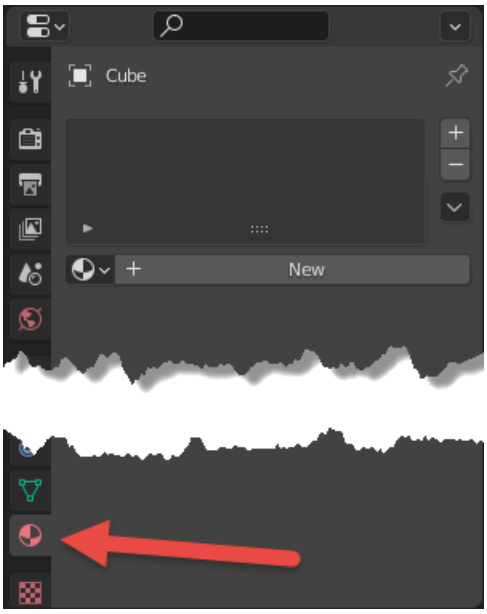


Fig 19. Material Properties option.

 - Click **New** to create a new material.

 - After that, you will see the new material appearing in the list of available materials to be linked to the object (), the properties of this material displayed in the **Material Properties** tab, and this material added to the material slot.

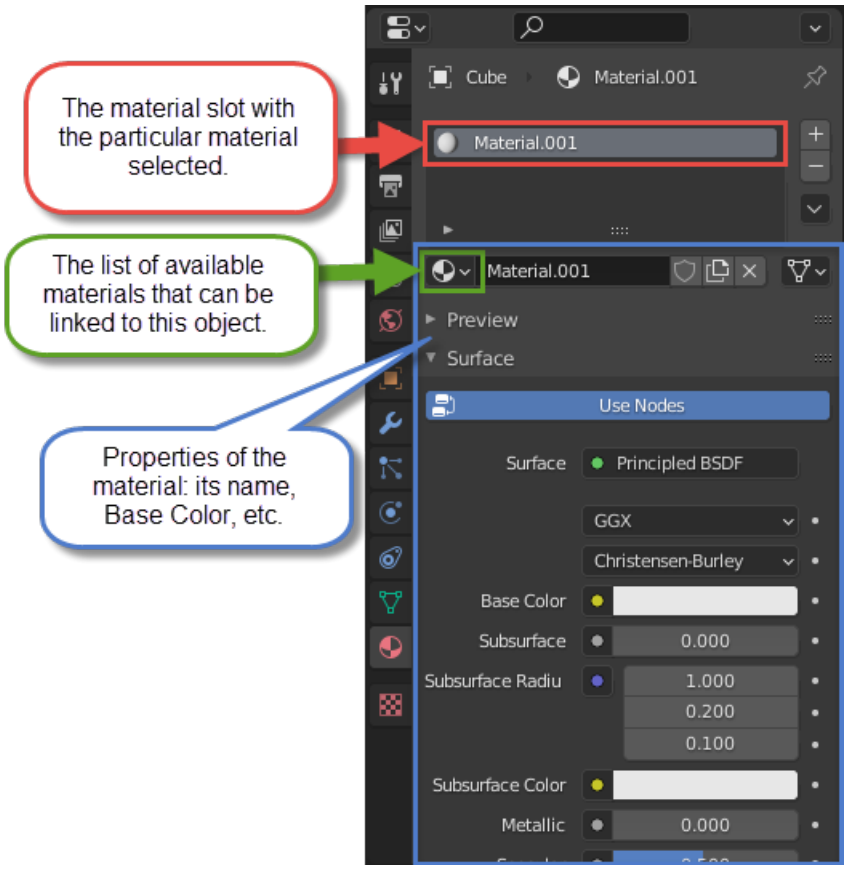


Fig 20. Material Properties tab.

- Change the properties of the material:
 - Double click its name and change it to "**+sky**". As we stated above, the exact spelling of this name (with plus) is very important.

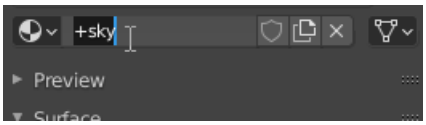


Fig 21. Name the material "+sky".

 - Since you probably want to distinguish this material visually in Blender, change the **Base Color** of this material by clicking this field and using the standard color selector afterwards.



Fig 22. Setting the Base Color of the +sky material.

> **NOTE**
>
> Colors of materials in Blender do not affect the final colors in the game, they are used for your convenience only. E.g. in the engine, the color of the **+sky** material will be defined by the specific skybox you assign to it (if you do not assign it, it will be of the default blue color).

- Now, let's create and set up the material for the "floor" of the large cube. Let's call this material "**ground**". This name is not so strictly defined as "**+sky**".

> **NOTE**
>
> We are creating a *static geometry* of the level (i.e. the geometry that will not change during the gameplay: the ground, sky, hills, rocks, etc.), and, since we will export it to the ASS file from the resulting FBX, our modding tools and engine will know that this is the static geometry. Due to that, this geometry will have the collision properties by default, and the player will not be able to fall through the ground for example. In fact, the player will not be able to leave the limiting static geometry of the level in any case, even when there is a **+sky** material assigned to it.

> **WARNING**
>
> To maintain the existing naming convention of materials (and their shaders, see Step #5), we recommend you to name your materials in Blender in snake_case (e.g. "**ground**", not "**Ground**"; and "**my_material**", not "**my material**").

So, let's create the **ground** material:

- While keeping the **Cube** node selected, click the plus button in the material slots panel. After that, the second material slot will appear.

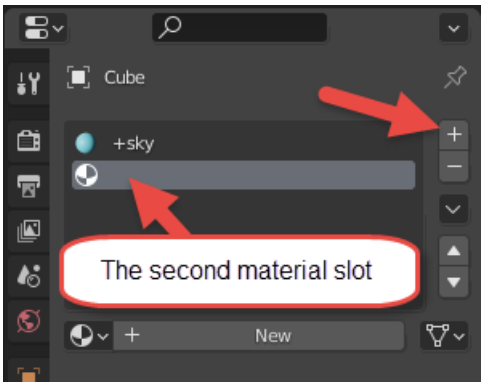


Fig 22. Add a new material below the +sky material.

- Click **New** to create a new material.
- When the new material appears, change its name to "**ground**" and its **Base Color** to green, similarly to the steps above. As before, the new material will automatically appear in the second material slot.

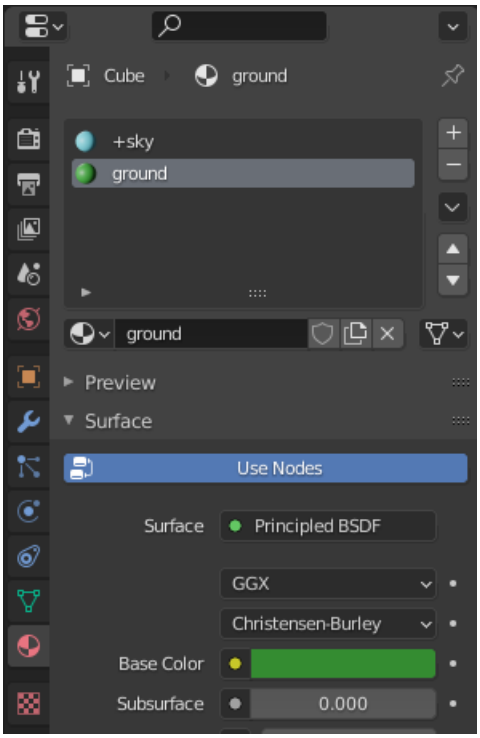


Fig 23. Setting the Base Color of the ground material.

- Now the materials are ready and we can assign them to different parts of the large cube. In the large cube, we will assign **+sky** to the faces of "ceiling" and "walls" and **ground** to its "floor" face.

 To do this:

 - Switch to the **Edit Mode**.

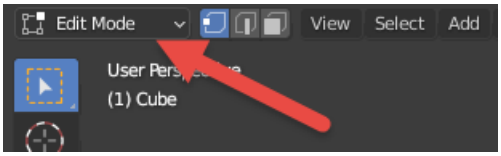


Fig 24. The edit more drop down in the Blender UI.

 - In the **Edit Mode**, next to the mode selection drop-down, 3 small buttons allow you to switch between the selection of vertices, edges, and faces. Select the selection of faces there (**Face select**).

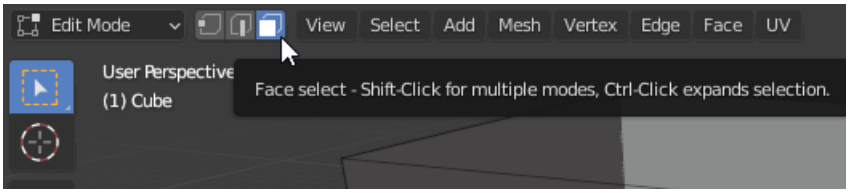


Fig 25. Face Select option.

 - Now, by clicking the particular face, you can select it (see below). And, by holding CTRL while clicking, you can select multiple faces.

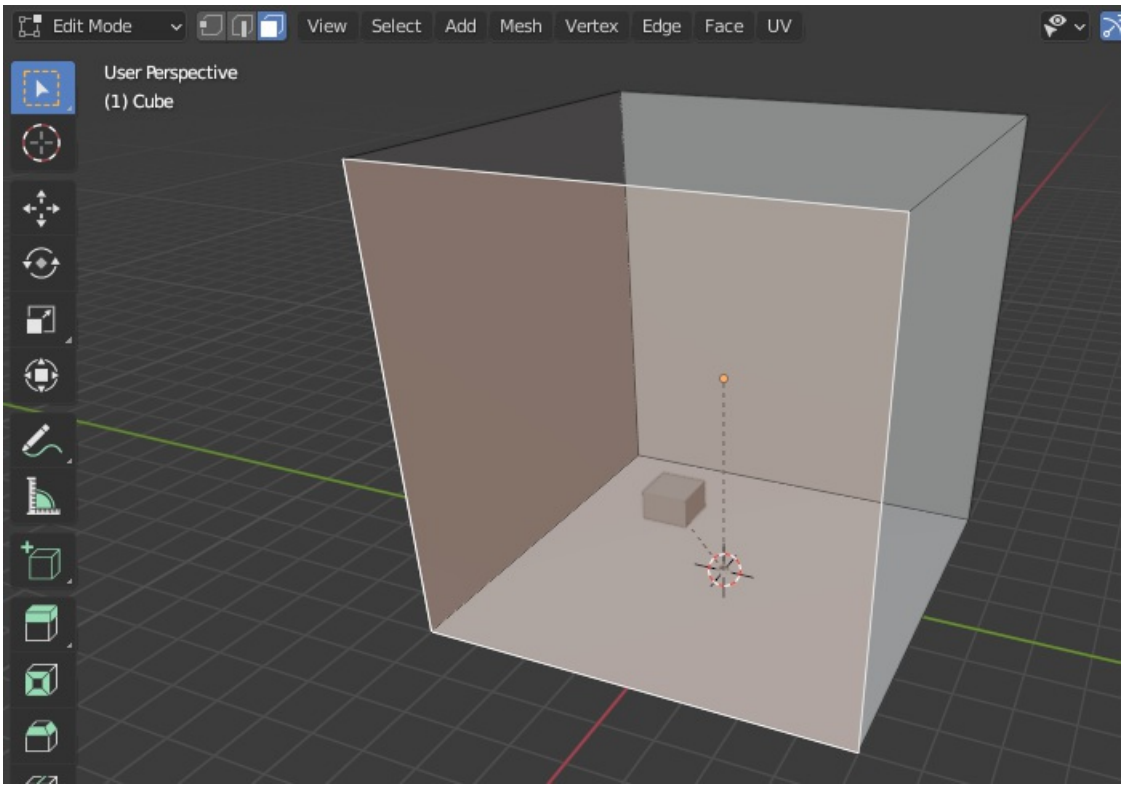


Fig 26. View of multiple faces selected.

- However, by default, you *cannot* select the faces that are behind the other geometry, i.e. the two back walls of our cube.

Of course, you can select them by constantly rotating the cube, but doing this can make a person a bit dizzy. Instead, you can use the **X-Ray** feature of Blender to select them:

- In the top right corner of the 3D Viewport, locate the **Toggle X-Ray** button () and click it to enable the X-Ray mode.

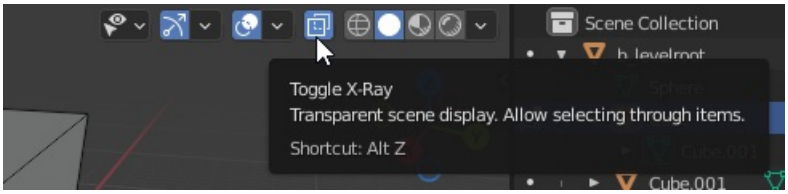


Fig 27. Toggle X-Ray button.

- With enabled X-Ray, you can select faces that are behind other geometry. So, now you can select the two back walls of our cube by clicking them while holding CTRL for multi-select.

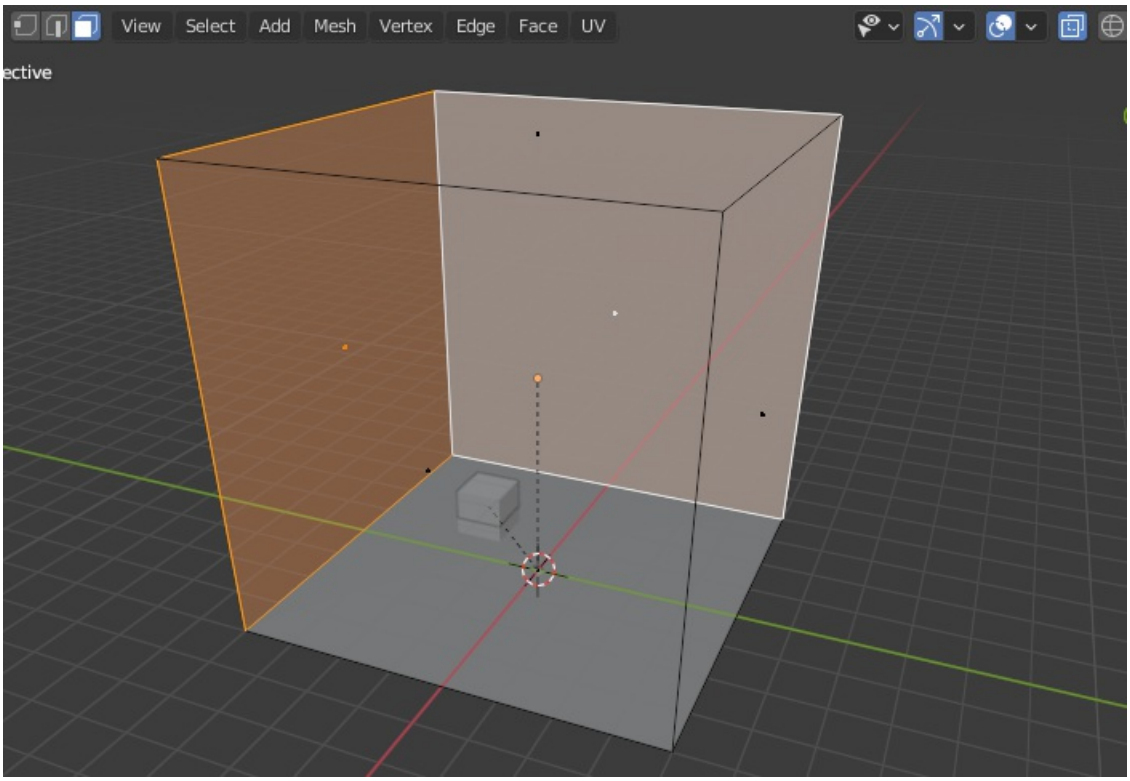


Fig 28. Back walls are selected with x-ray.

- Now, select the front "walls" and "ceiling":
 - While keeping the back "walls" selected, click **Toggle X-Ray** one more time to disable X-Ray.
 - Then, while holding CTRL, select the faces of the front "walls" and "ceiling":

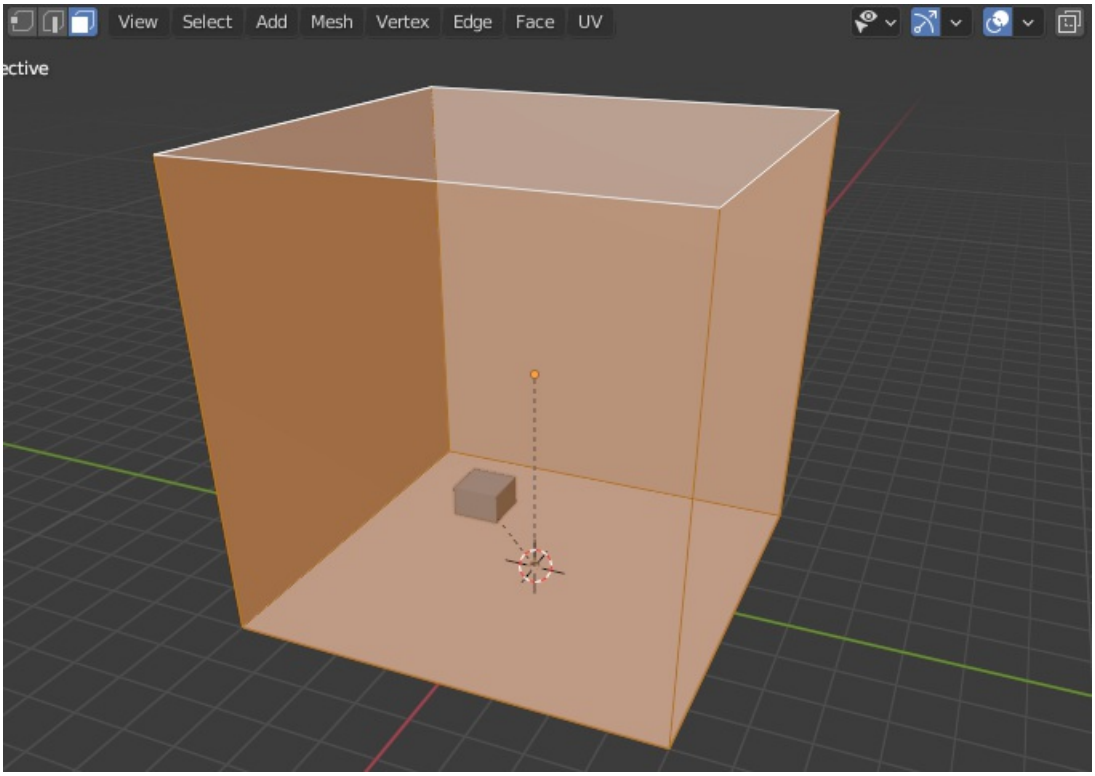


Fig 29. Front "wall" and "ceiling" selected.

- Time to assign the **+sky** material to these faces:
 - While keeping the target faces selected, in the **Properties** panel (with the **Material Properties** tab still opened), select the material slot with the **+sky** material (see below).

- Click **Assign** in this panel.

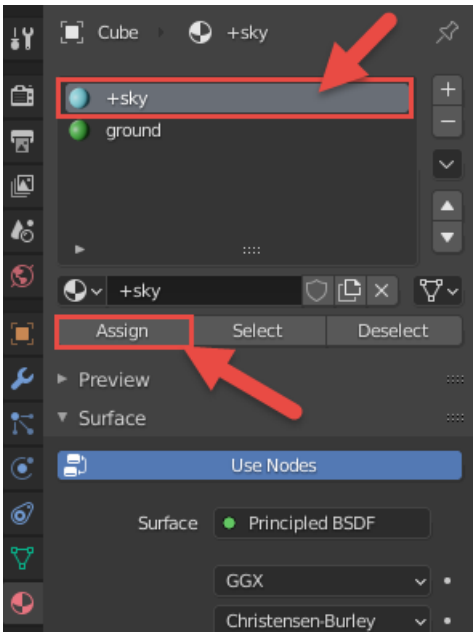


Fig 30. View of the Assign button with the +sky material selected.

- To ensure that this material has been assigned, switch to the **Material Preview** method () of **Viewport Shading** in the top right corner of the **3D Viewport** (see below). After that, your large cube will be displayed in blue.

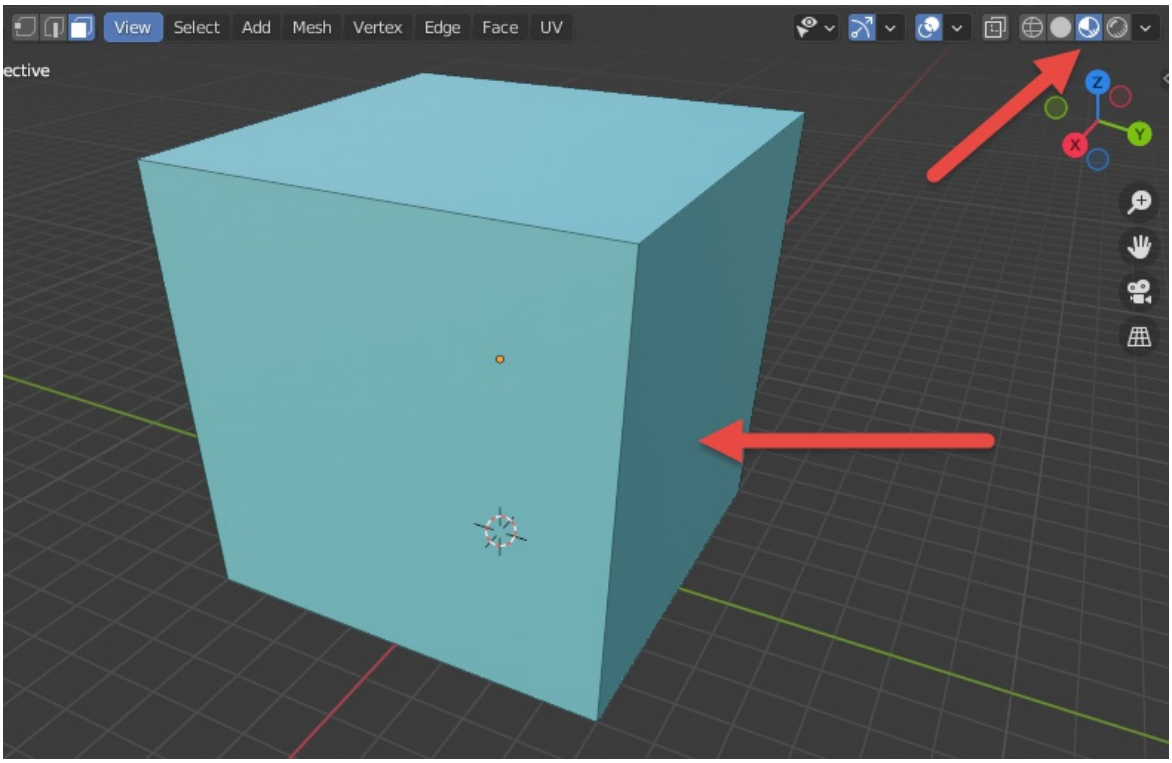


Fig 31. The large Cube with the +sky material assigned.

- Using the same technique, you can assign the **ground** material too:
 - Enable the X-Ray mode.
 - Select the "floor" face of the larger cube.
 - While keeping this face selected, in the **Properties** panel, select the material slot with the **ground** material and click **Assign**.

- To ensure that this material was assigned correctly, zoom in to get inside the large cube (use the mouse wheel for zooming). Inside, you should see that the walls and the ceiling are of the **+sky** color, and the floor is of the **ground** color (to rotate the view, hold the mouse wheel while moving the mouse). The small cube and the sphere of the frame node will be grey since we have not assigned our materials to them.

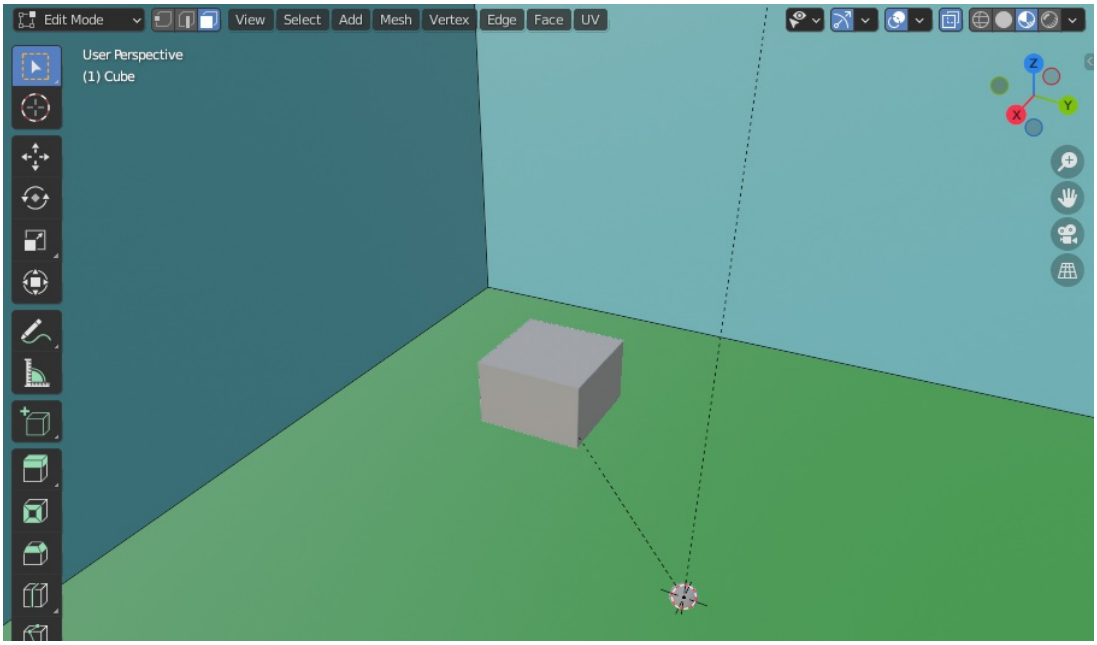


Fig 32. ground and +sky materials assigned to the correct faces.

> **NOTE**
>
> If you forget to assign materials to some part of the scene, objects without them will still be static geometry (with collisions), but will cause the **invalid_material** errors.

- Now, assign the materials to the small cube. The frame node can live without the material since it will be just a node in the hierarchy and will not be visible as a part of the level. Since we have already created our materials, this can be done simply:
 - Switch to the **Object Mode**.

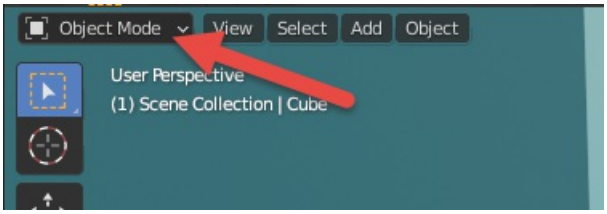


Fig 33. The Object Mode drop down.

 - Select the small cube.
 - In the **Properties** panel (with the **Material Properties** tab still opened), in the list of available materials to be linked to object (), select necessary material (**ground**). There is no need to select faces and click **Assign** in this case, since we are assigning a single material to the whole object.

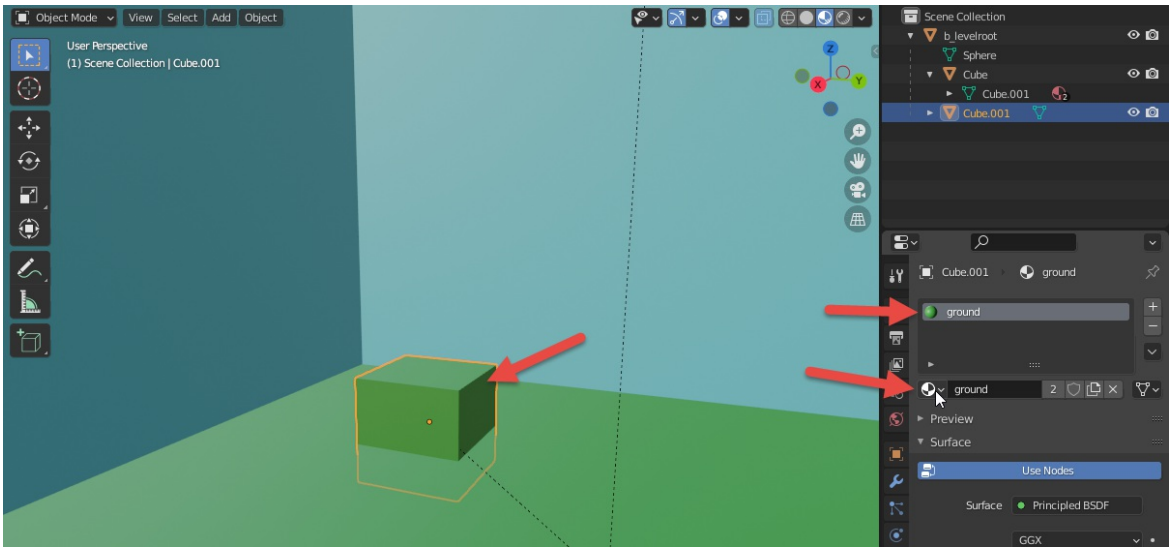


Fig 34. Assigning the ground material to the small Cube.

10. Materials are assigned, time to add some lighting to the scene.

 For illustrative purposes, let's add a single light spot located above the small cube we have added within the large one:

 - In the **Object Mode**, enable the **Cursor** tool by clicking it in the Tools panel located at the left side of the **3D Viewport**.

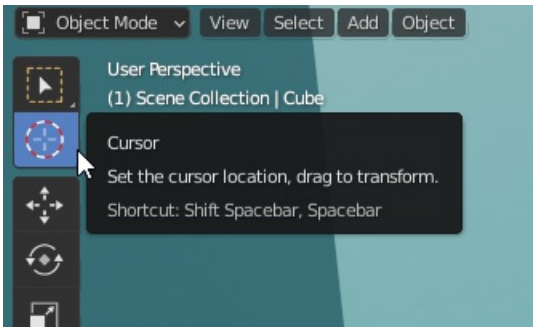


Fig 35. The Cursor Tool.

Using this tool, you will be able to change the location of the cursor in **3D Viewport** by clicking on any part of the screen (see official help for details). However, by default, when you use this tool without orthogonal views, the cursor is placed on the geometry you click (i.e. the *depth* of the location of the cursor is defined by it).

- While you are inside of the large cube (if not, zoom in into it), click any point on the small cube to place a cursor on it.

> **NOTE**
>
> To select the target point, you may want to rotate or move your view. To rotate, hold the mouse wheel while moving the mouse. To move, hold both SHIFT and mouse wheel while moving the mouse.

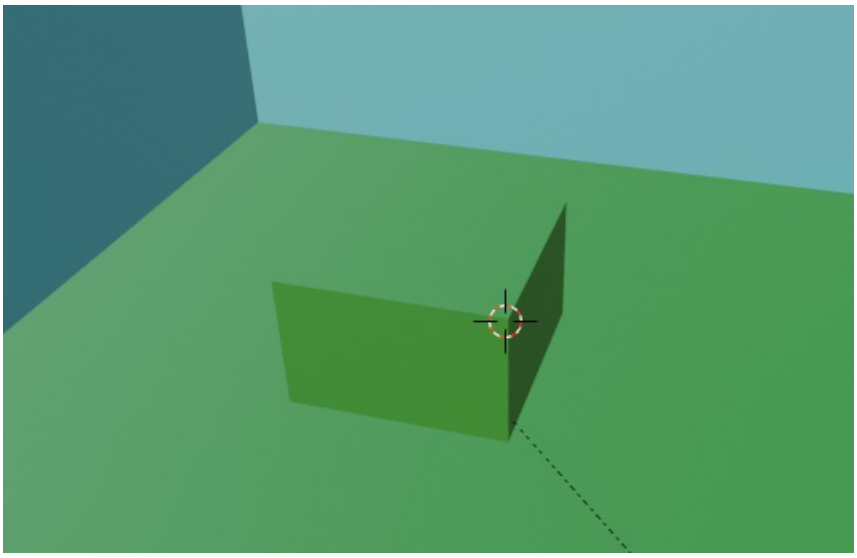


Fig 36. View of the small cube with a cursor on it.

- Now, add the light spot to the cursor location by selecting **Add** > **Light** > **Spot** in the **Add** menu.

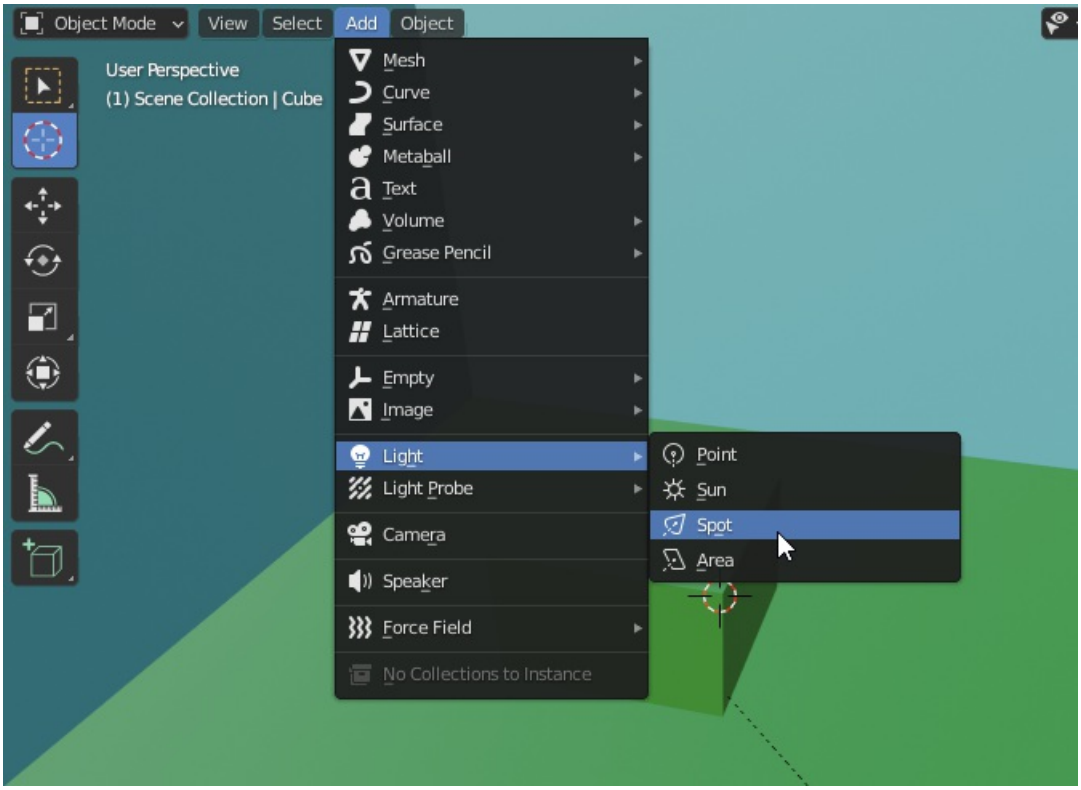


Fig 37. Light > Spot option within the Add menu.

- After that, the new light node will appear in the **Outliner** and the corresponding light object in the **3D Viewport** (in the location of the cursor).

- Select the **Move** tool () to move it where necessary (using arrows, as we did before) or, alternatively, press **G** to activate "Move" transformation mode and select the exact location for the spot. For example, you can place the light spot just above the small cube.

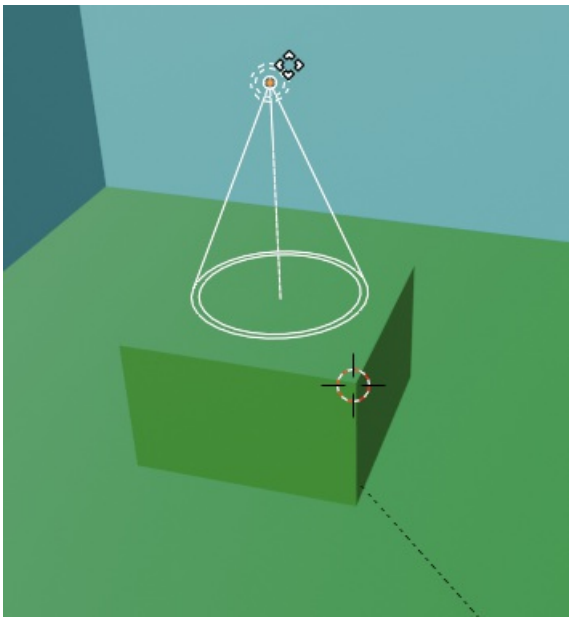


Fig 38. New spot light placed above the small cub.

- As with other objects, you also need to parent it to the frame node.To do this: in the **Outliner**, select the light spot node, then, while keeping it selected, select the frame node, then, in the **3D Viewport**, right-click any point of the screen and select **Parent > Object**. For details, see step 6 above.

After doing this, the light spot will be also below the **b_levelroot** in the hierarchy.

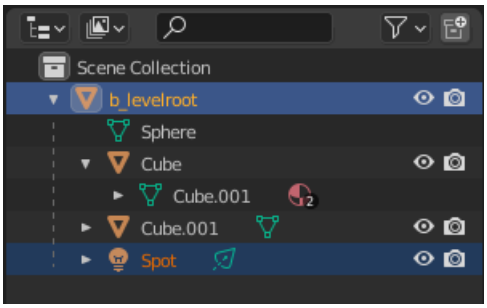


Fig 39. Hierarchy with the Spot light parented to the b_levelroot.

- In the process of working with lights, it is useful to preview them in the scene. To do that, you can switch to the **Rendered** mode of the **Viewport Shading** in the top right corner of the viewport.

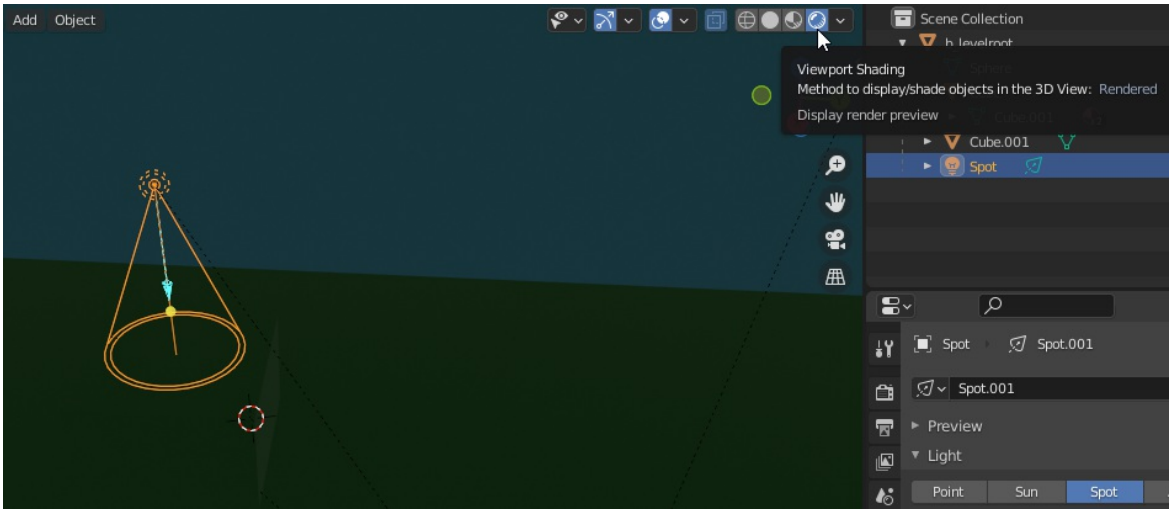


Fig 40. Viewport Shading option.

- As you can see in the preview, the scene is dark and the light source is not visible.

> **NOTE**
>
> The power of lighting in the viewport of Blender and the game engine is different, you should take this into account.

- However, let's adjust some of its properties to make it visible (with other options you can play on your own):
 - If not already selected, select the light spot node.
 - In the **Properties** panel, in the **Object Data Properties** tab, modify the following properties of the light spot:
 - **Power** – the power of the light source. By default, it is **10** Watt, which is very low. Let's set **300** Watt here.

 > **NOTE**
 >
 > 300 Watt will still look a little bit dark in Blender, but in the game engine it will look different.

 - **Size** – as you can see, the light spot is producing a cone of light. The **Size** parameter controls the angle of this cone, in degrees. By default, it is **45** degrees, which is a bit narrow for us. Let's make it **90** degrees.

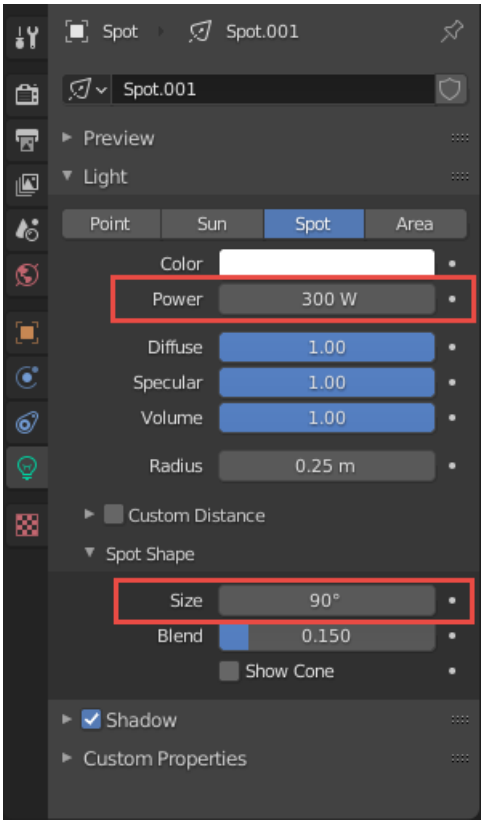


Fig 41. Power and Size options within the Spot Light Properties.

 - As you can see in the **Rendered** mode of the **Viewport Shading**, now the light from the light spot is visible at least. Please don't mind that it still looks dark now.

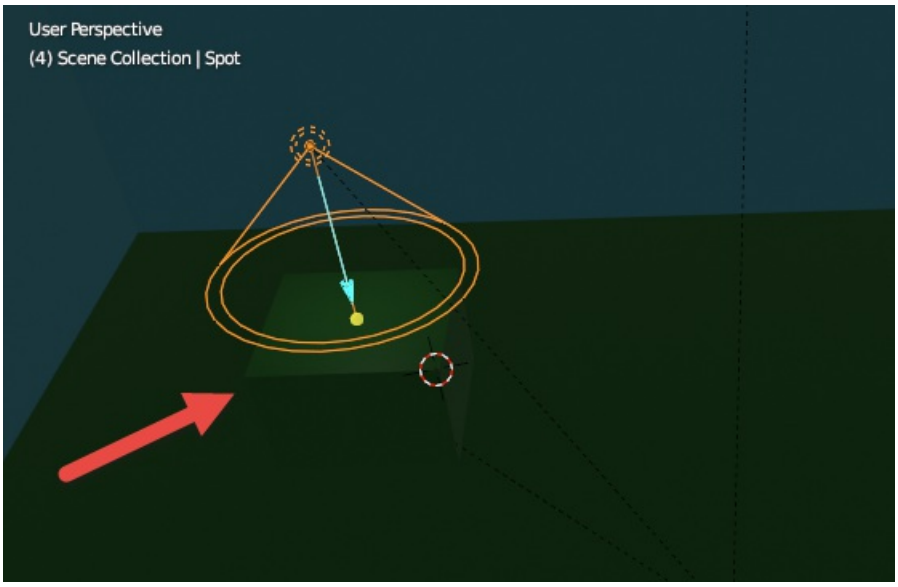


Fig 42. View of the spot light properties within the Blender viewport.

11. Now it is a good idea to save the final version of your source Blender scene. The saving itself is done absolutely in a standard way (**File > Save/Save as** in the main menu of Blender). However, the location where you will save the scene is important. Actually, it is most important not for the Blender scene file, but for the FBX file exported from it (see the next section). But, it is a good practice to store the initial source file of the scene and FBX in one folder anyway.

So, let's save your source scene to the folder of your future FBX.

To do this:

- Open Windows Explorer, and, in the Halo 3 Editing Kit root folder, open **\data\levels** subfolder in it.
- In **...\data\levels**, create a subfolder for mods of your levels, e.g. **mod_levels**
- In this folder, create a separate folder for your level, e.g. **my_level_1**
- In **...\data\levels\mod_levels\my_level_1** folder, create a **structure** subfolder.
- In Blender, save your source scene in the **...\data\levels\mod_levels\my_level_1\structure** folder with the same name as the name of the level. E.g. as **my_level_1.blend**.

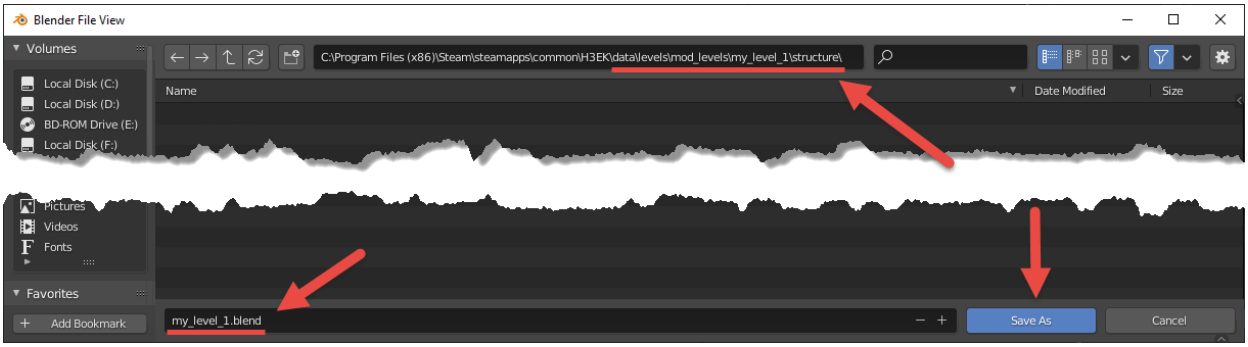


Fig 43. Folder Location and File Name for saving.

> **NOTE**
>
> Actually, the name of the initial **.blend** file is not important, but it is convenient to have it named similarly to the FBX file for easier identification.

12. Now, you saved your work and can proceed to the next step – Exporting Geometry to FBX.

Quick Start Process Step 2 - Exporting Geometry to FBX

11/23/2022 • 6 minutes to read

In this step, we will export our scene from Blender to the FBX format.

In Halo 3 modding, this format is used as an intermediate format for all levels and models. If you are using Maya or 3ds Max, you will need to export the initial scene to this format too. However, in this guide, we will describe only export from Blender.

The general pipeline for exporting to FBX from Blender is very simple (see screenshots below). You need to:

1. Open **File > Export > FBX (.fbx)** from the main menu of Blender.
2. Specify the name/location of the target FBX file to be created and the export settings in the appearing dialog.
3. Click **Export FBX** to create the FBX file.

However, some aspects are important if you want to create a correct FBX, see subsections below for particular instructions.

> **WARNING**
> If you simply perform export without specifying correct settings as described in the subsections below, this will not work well and will almost certainly result in the incorrect FBX file. After exporting with the correct settings, you will have the FBX file in the appropriate location.

In our case, the file will be the following:

...\data\levels\mod_levels\my_level_1\structure\my_level_1.fbx

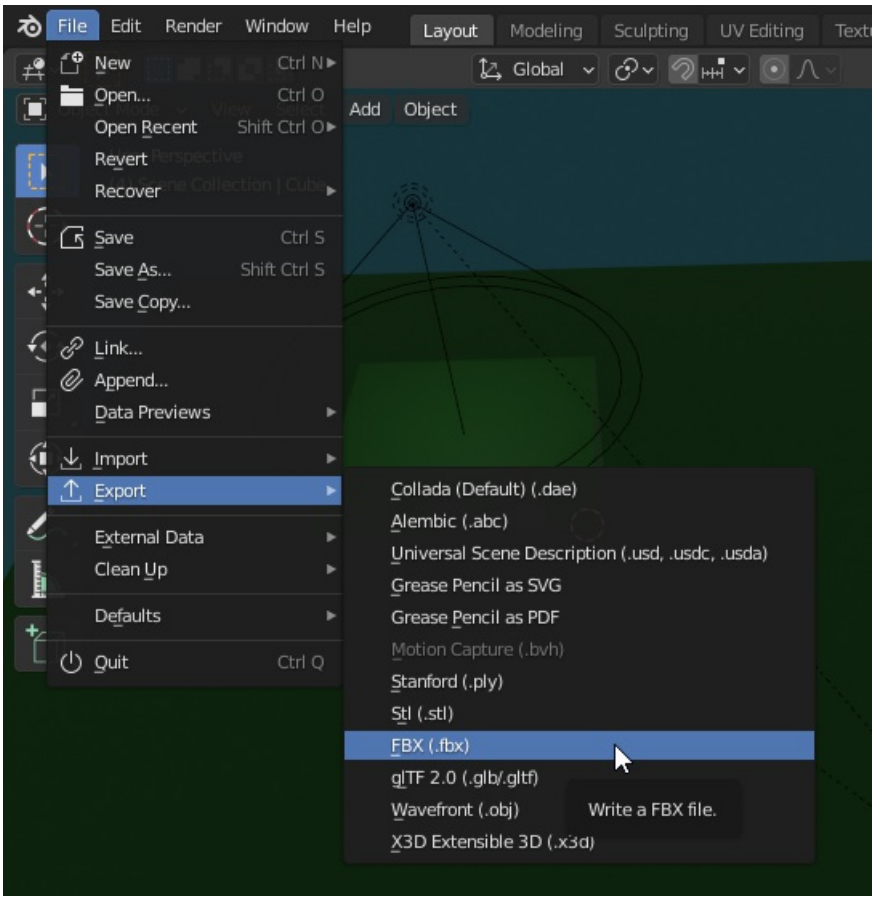


Fig 1. File > Export > FBX option when Exporting your meshes.

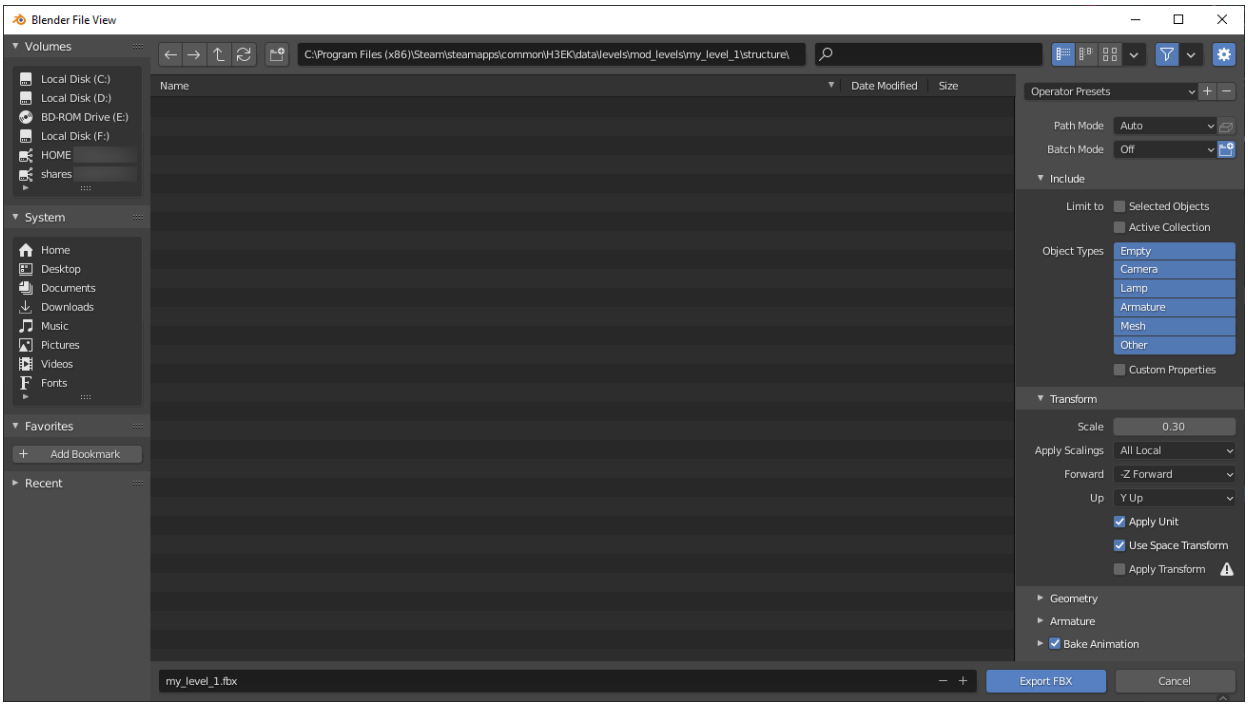


Fig 2. Export FBX Dialog.

Correct Location and Name of the FBX File

First of all, let's talk about the location of the target FBX file. You need to save your FBX to the **data** folder within the root folder of your Halo 3 Editing Kit directory.

> **NOTE**
>
> For general details on the **data** and **tags** folders, please refer to the Quick Start Overview doc.

Particularly, you will need to save your FBX within the **...\data\levels** folder, within the **structure** subfolder of the folder of your level, which you need to create somewhere in **...\data\levels**.

For consistency and to keep all your level mods in one place, we recommend you to create a special folder for them in the **...\data\levels** folder. For example, you can name it **mod_levels**.

In this folder, you will need to create the folder of the level. E.g. **my_level_1**.

> **NOTE**
>
> For uniformity, we recommend the **snake_case** for the names of files and levels.

In this folder, you must create the folder named structure, which will be the home for your target FBX.

> **NOTE**
>
> You need to follow the strict folder structure for the files that participate in the modding process. Particularly, the folder structures of the **tags** and **data** folders must be similar for your custom files. This is necessary for the correct operation of the modding tools. Along with it, this will allow you to find your source files easily when you know their tag files and vice versa.

I.e., the final path to your FBX should be similar to the following:

...\data\levels\<folder_for_mods_of_levels>\<name_of_level>\structure

For example, for our FBX:

...\data\levels\mod_levels\my_level_1\structure

We recommend you to use the name of the level folder as the name of your FBX file. For uniformity, it is better when these names are the same. For example, you can specify **my_level_1.fbx** as the name of the target FBX file.

I.e., the name and location in the export to FBX dialog should be similar to the following:



Fig 3. File name and Location.

Correct Settings for Modelling: Directions of Axes, Triangulation, and Units

Directions of Axes While Modelling

During modeling, you need to take care of the orientation of the axes. Along with settings for the directions of

these axes during the export, this is necessary for your level to have the correct orientation in the game. The directions of axes while modelling should be the following:

- **X**-axis – should correspond to the **Forward** direction.
- **Y**-axis – should correspond to the **Leftwards** direction.
- **Z**-axis – should correspond to the **Up** direction.

For sure, you can rotate your viewport as you like during the modelling, but to take view what orientation your level will have, you can always change the directions of axes as described above to ensure that all is correct.

For example, for the level we have modeled in Step 1, in this orientation of the axes in Blender the level will look like the following, which is what we need:

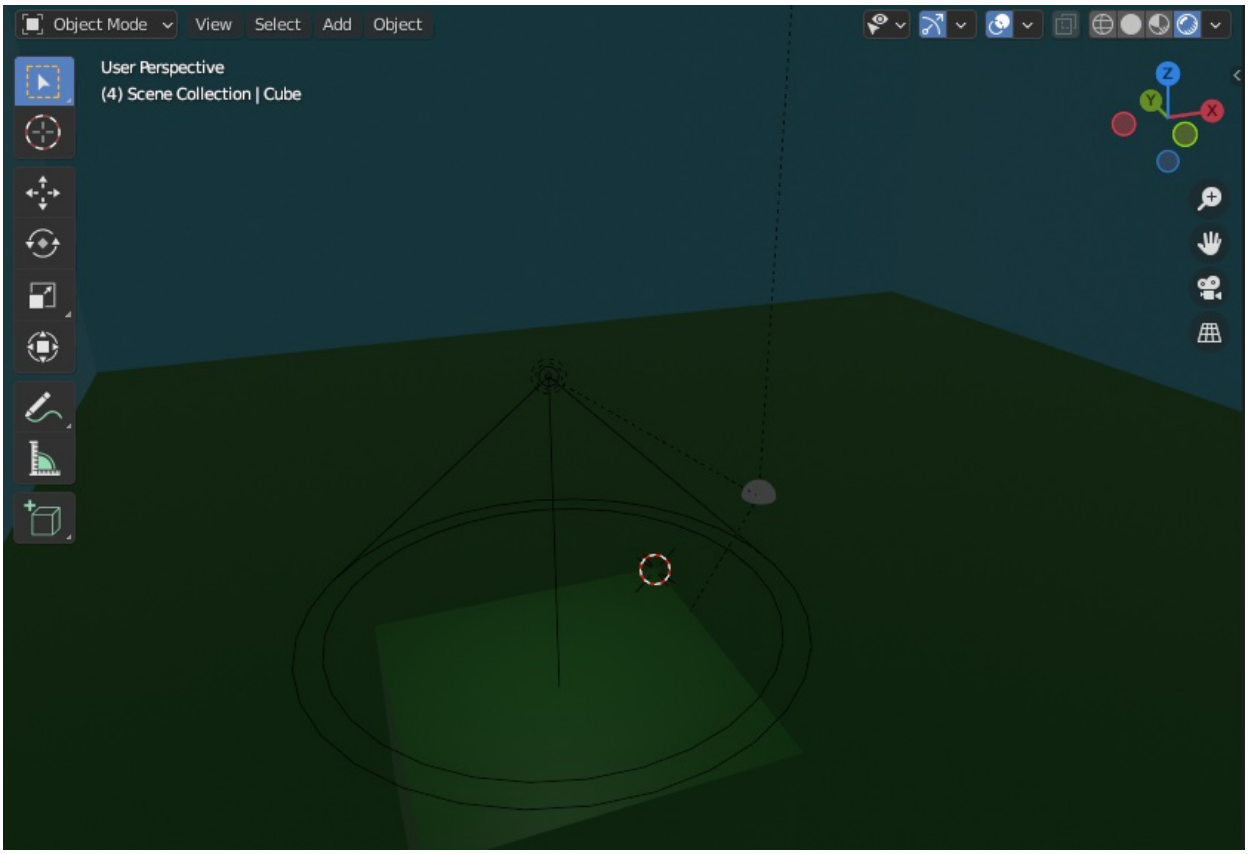


Fig 4. Axis Directions in Blender.

Triangulation

If you are modeling something complex and smooth, you need it triangulated. Otherwise, the level can cause errors when opened by the engine.

Blender has multiple ways to do it. Particularly, there is a **Triangulate Faces** tool (in the **Edit Mode**, select the target mesh or part of its faces, then open the Face menu and select **Triangulate Faces**, see official help) and, also, the **Triangulate Modifier** (see official help).

> **WARNING**
> Unfortunately, the triangulation of complex and smooth objects is a mandatory requirement. If you still want to make some surfaces as smooth as possible, use a larger number of triangles for them.

However, our simple level does not need this procedure, so we will just mention this as a starting point for your own experiments.

Units

Along with other things, you need to take care of the unit system and units you use when you are modeling.

This is important since, depending on the units you use, your export settings (particularly, the **Scale**, see below) can vary – to obtain the correct dimensions of objects in the engine.

In this tutorial, we are using the **Metric** unit system and **Meters** for length, and we will need to apply a conversion coefficient for **Scale** in the export settings (see below).

In Blender, unit properties of the scene can be changed in the **Properties** panel > **Scene Properties** tab > **Units** section (see official help for details).

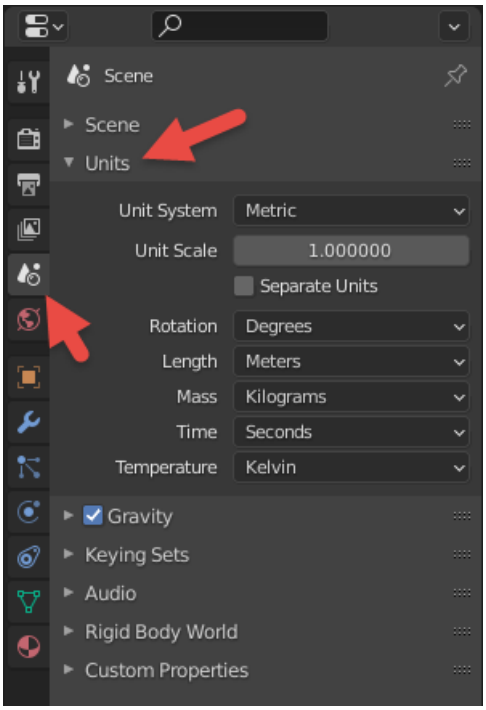


Fig 5. Units in the Scene Properties.

Correct FBX Export Settings

When you open the export dialog, export settings for the transformation of the scene into FBX are displayed on the right side of the dialog.

If you have followed the previous steps of the tutorial, have a correct orientation of axes (see above), and have used a **Metric** system / **Meters** while modeling (see above), these settings will need to be the following:

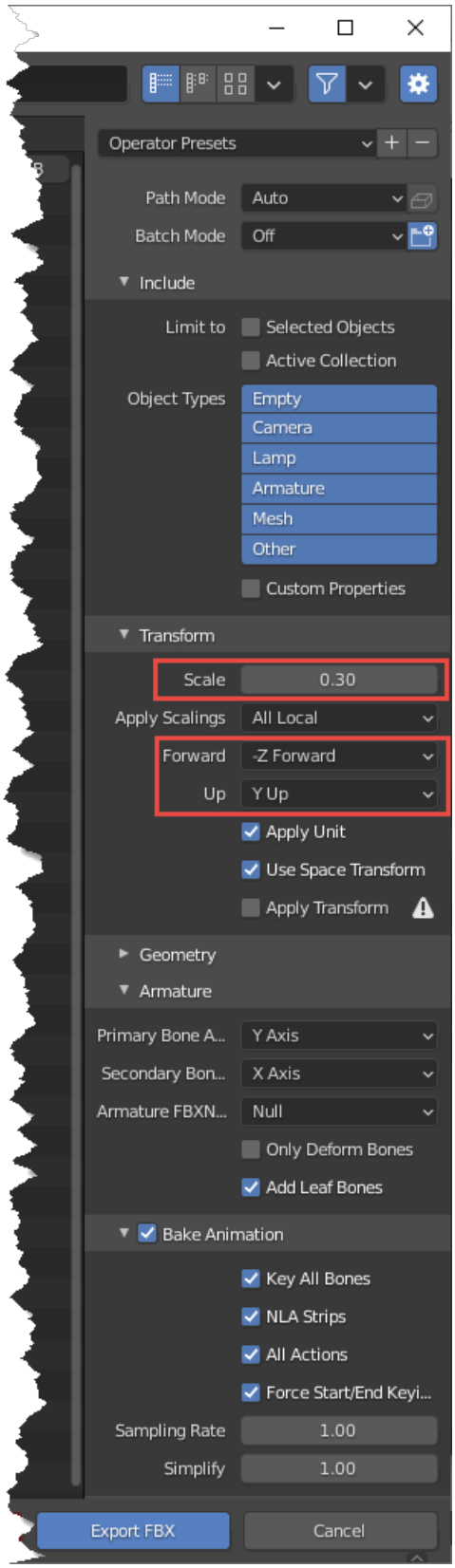


Fig 6. View of the correct FBX export settings.

All export parameters should be set as displayed in the picture above.

> **IMPORTANT**
>
> Please note that the correct **Scale** specified in these settings depends on the units system (length units) that were used during modeling.

Particularly, in this tutorial, we have used the **Metric** system and **Meters** while modeling. We recommend you

to use the same unit system for passing this tutorial also.

Halo 3 uses its own units system. For simplicity, if you model in **Meters**, you can consider that 1 Halo length unit corresponds to **0.3048** of a meter. Because of that, we have specified **0.30** as the value of the **Scale** parameter.

Along with that, as you see, we have also performed a conversion for axes (have set **Forward** to "**-Z Forward**" and **Up** to "**Y Up**"). This conversion is valid since while modelling we had the orientation of axes as described in **Direction of Axes While Modelling** above.

After exporting initial scene from Blender to FBX with correct parameters. you can proceed to the next step – Conversion from FBX to ASS.

Quick Start Process Step 3 - Conversion from FBX to ASS

11/23/2022 • 2 minutes to read

If you have followed Step #1 and Step #2 of this guide, you should now have two files in the ... \data\levels\mod_levels\my_level_1\structure directory (or at a similar path): the .blend file of the source Blender scene and the .fbx file that you have exported from it.

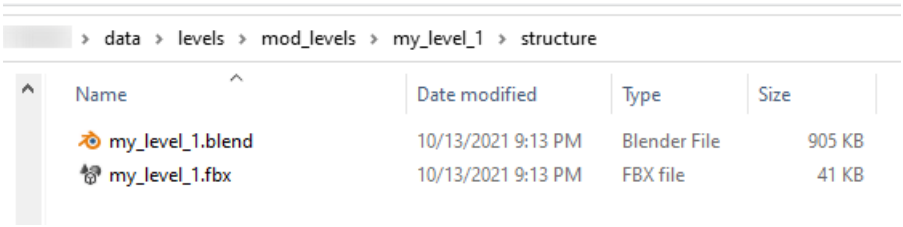


Fig 1. Initial files created in the previous tutorial steps.

Now, you will need to convert the FBX file to the ASS format (static geometry of the level).

This can be done using the **tool_fast.exe** tool.

> **NOTE**
> For general info on this tool, see the Quick Start Overview doc.

Particularly you will need to execute the **fbx-to-ass** command using **tool_fast.exe**:

```
tool_fast.exe fbx-to-ass <source fbx file> <target ass file>
```

For example:

```
tool_fast.exe fbx-to-ass "C:\Program Files
(x86)\Steam\steamapps\common\H3EK\data\levels\mod_levels\my_level_1\structure\my_level_1.fbx" "C:\Program
Files (x86)\Steam\steamapps\common\H3EK\data\levels\mod_levels\my_level_1\structure\my_level_1.ass"
```

> **WARNING**
> When the paths to your files within commands contain spaces in the name(s) of the directory(ies), you need to put them in double quotes (e.g. "the path\to\file"), like in the example above. This is necessary for the correct execution of the command.

You will need to execute this command in the command line of the **Command Prompt** from the root directory of Halo 3 Editing Kit (where the **tool_fast.exe** executable is located).

To open the Command Prompt in this directory, you can do the following:

1. Open the the root directory of Halo 3 Editing Kit.
2. Click the path to the directory (displayed at the upper bar, above the folders) to make it editable.

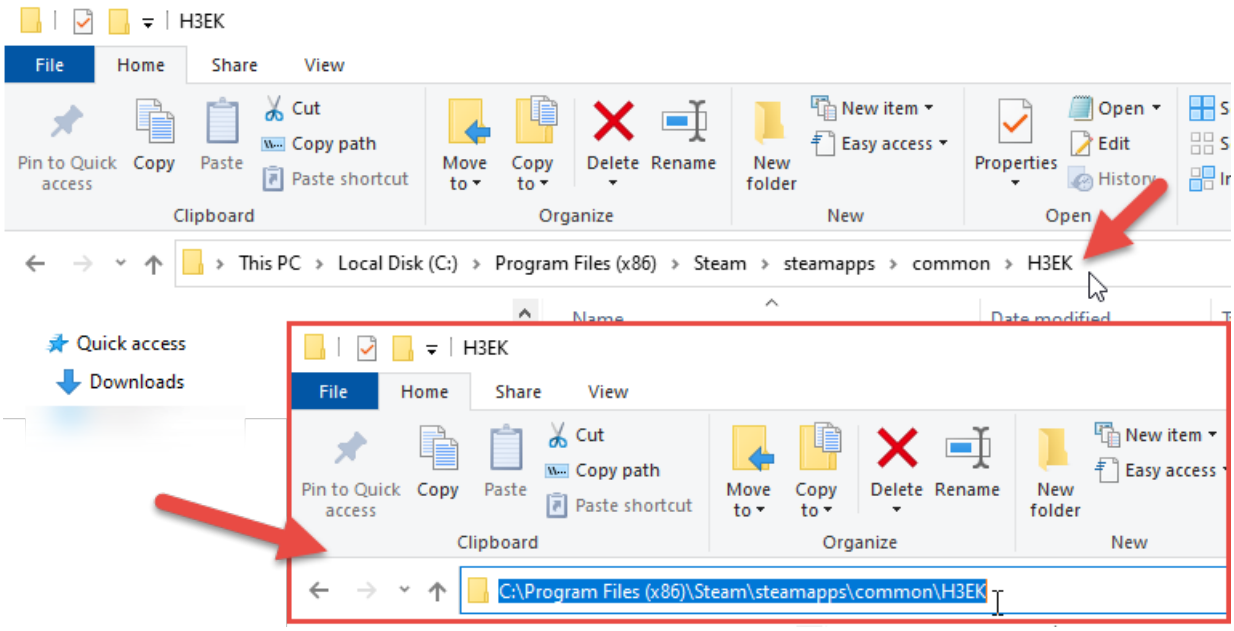


Fig 2. Highlight the folder path to change the text.

3. Replace this text with the **cmd.exe** command and press ENTER to execute it.

View of the folder path field now populated with cmd.exe with a text box prompting to press enter.

Fig 3. Enter cmd.exe and then press Enter

4. After that, the new command prompt window will open and will wait for your input. Moreover, it will be opened exactly in the the root directory of Halo 3 Editing Kit, as you can see in the sceenshot below.

View of the command prompt window open with the directory set to the folder where you Halo 3 Editing Kit is installed.

Fig 4. Command Prompt with the root directory set to the H3EK.

After that, you will be able to execute the **fbx-to-ass** command of **tool_fast.exe** as mentioned above.

After execution of this command, the tool may display a warning or two, but you can ignore them.

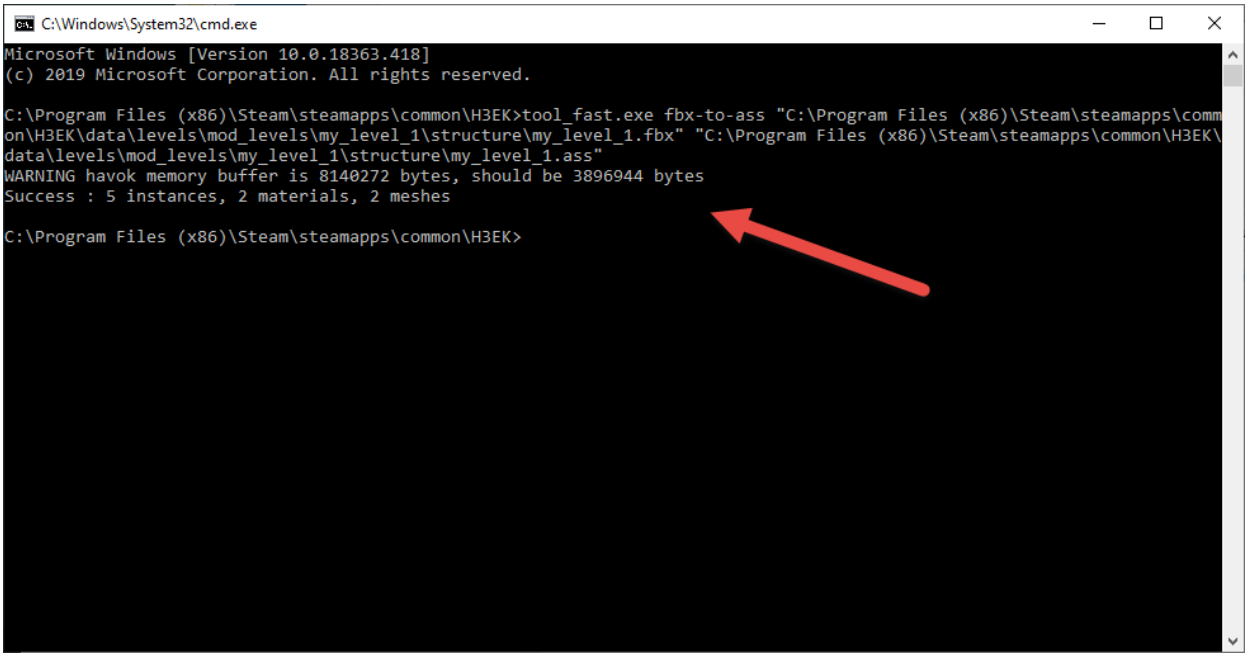


Fig 5. Output in tool_fast.exe after executing the fbx-to-ass command.

After successful execution of this command, the new **.ass** file will appear at the **structure** folder of your level. In

our case, it will be **my_level_1.ass** file:

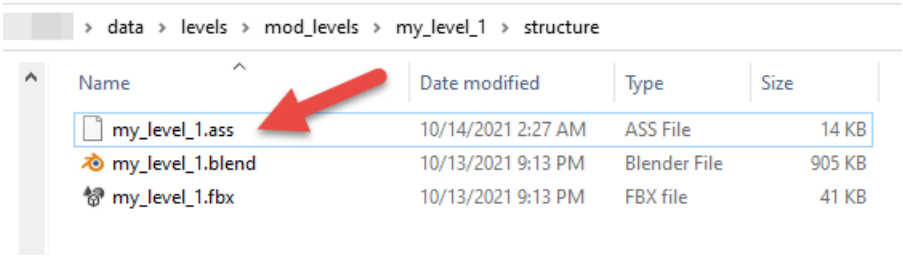


Fig 6. File output once the fbx-to-ass command is completed.

After that, you can proceed to the Creation of Initial Tag Files step.

Quick Start Process Step 4 - Creation of Initial Tag Files

11/23/2022 • 2 minutes to read

After the creation of the ASS file, you can start building the Tag files of your future level.

> **NOTE**
> For general info on Tag files, see the Halo 3 Modding - General Concept doc.

Particularly, you can use the **structure** command of the **tool_fast.exe** tool to build the main scenario tag files (the **.scenario** tag file and others) of the level for Sapien from the **ASS** file.

The format of this command is the following:

```
tool_fast.exe structure <path to the .ass file relative to the "data" folder>
```

> **NOTE**
> The path to the **.ass** file in this command is specified relative to the **data** folder. I.e., if the full path to the target file is "**C:\Program Files (x86)\Steam\steamapps\common\H3EK\data\levels\mod_levels\my_level_1\structure\my_level_1.ass**", you will need to specify **levels\mod_levels\my_level_1\structure\my_level_1.ass** as the path to the .ass file as the parameter of this command.

For example:

```
tool_fast.exe structure levels\mod_levels\my_level_1\structure\my_level_1.ass
```

After the execution of this command, the tool will display a few errors and warnings, but you can ignore them again.

View of the command prompt output when running the structure command in the tool_fast.exe.

Fig 1. The command prompt output in tool_fast.exe when running the structure command.

Most importantly, this will command will generate three scenario tag files:

- **.scenario** – This contains most of your level's settings, such as which sky your level has, what weapons and items appear and where, what locations players spawn in, and so on. This is what you'll be editing in Sapien when you place weapons, items, scenery objects, etc.
- **.scenario_structure_bsp** – This is the actual architecture of your level. The geometry you export from the 3D Modelling tool eventually ends up here. Notice that this is separate from the .scenario file, which means that updating your level's geometry doesn't affect anything else you've placed within the level. Be careful when you move walls or modify your level's architecture that you find and fix any items that might end up outside the world or stuck inside a wall.
- **.scenario_structure_lightmap** – Eventually you'll want to light your level. This can take a long time to

compute, but it's definitely needed if you want a truly finished product. When you do, you'll also have a **scenario_structure_lightmap** that contains all that lighting information. The **scenario_structure_lightmap** itself also requires a **.bitmap** file that's created alongside it.

These files will be located in the folder of your level in the ***tags*** directory. This folder will be created automatically.

For example, if your target ASS file was located at:

C:\Program Files (x86)\Steam\steamapps\common\H3EK\data\levels\mod_levels\my_level_1\structure\my_level_1.ass

then the generated scenario tag files will be located at:

C:\Program Files (x86)\Steam\steamapps\common\H3EK\tags\levels\mod_levels\my_level_1

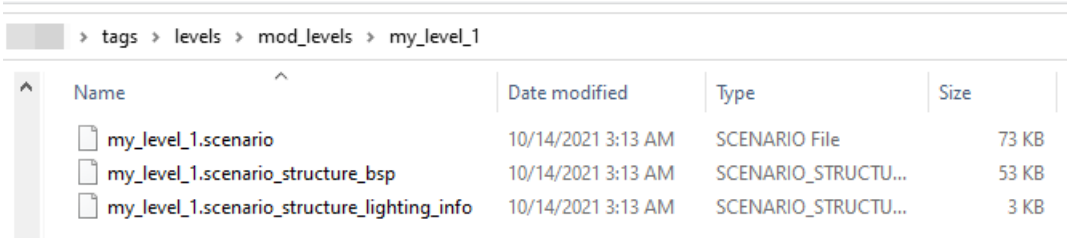


Fig 2. New files output by the structure command.

You will need these files at the next steps of this guide.

Now, you can proceed to the Modification of Tags in Guerilla step.

> **NOTE**
> Moreover, if you are eager to view the geometry of your level even without lighting and textures, you can simply open the generated **.scenario** tag file (e.g. **my_level_1.scenario**) using Sapien.

Quick Start Process Step 5 - Modification of Tags in Guerilla

11/23/2022 • 6 minutes to read

Now, you are ready to modify some properties of your level in Guerilla. Guerilla works with Tag files of the game/level and allows you to modify them.

> **NOTE**
>
> For general info on Tag files, see the Quick Start Overview doc.

When you first open **guerilla.exe**, which is located at the root directory of Halo 3 Editing Kit, you will see the contents of the **tags** folder in the left navigation panel. This folder contains all tags of the game, nicely categorized by folder hierarchy.

Since you have already converted your level into tags, it contains the tags of your level too. So you can open, view, and modify them.

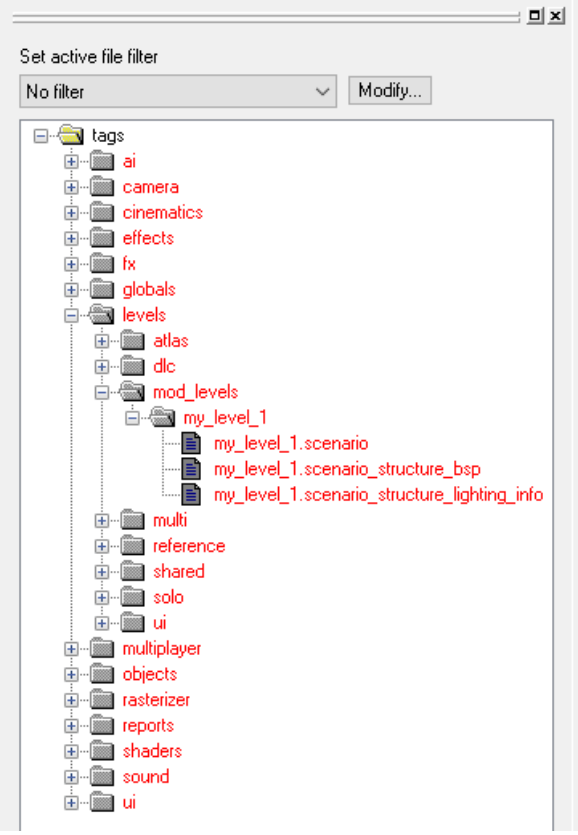


Fig 1. View of the Guerilla navigation.

Currently, your level lacks textures, so let's assign them first. Roughly speaking, we will need two textures, one for the **+sky** material (it's the special case) and one for the **ground** material, but we will use the provided textures for simplicity.

> **NOTE**
>
> Creating a custom sky and the full process of assigning or creating a texture (shader) is out of the scope of this basic tutorial.

Selection of Sky

Let's begin with the **+sky**:

1. In Guerilla, in the left navigation panel, expand the hierarchy and locate the **.scenario** file of your level (see the picture above). E.g. **my_level_1.scenario**
2. Double-click it and wait a little bit until it opens.
3. You will see a vast dialog with a lot of sections and properties. This dialog displays the properties of your level. (Its full description is obviously out of scope.)

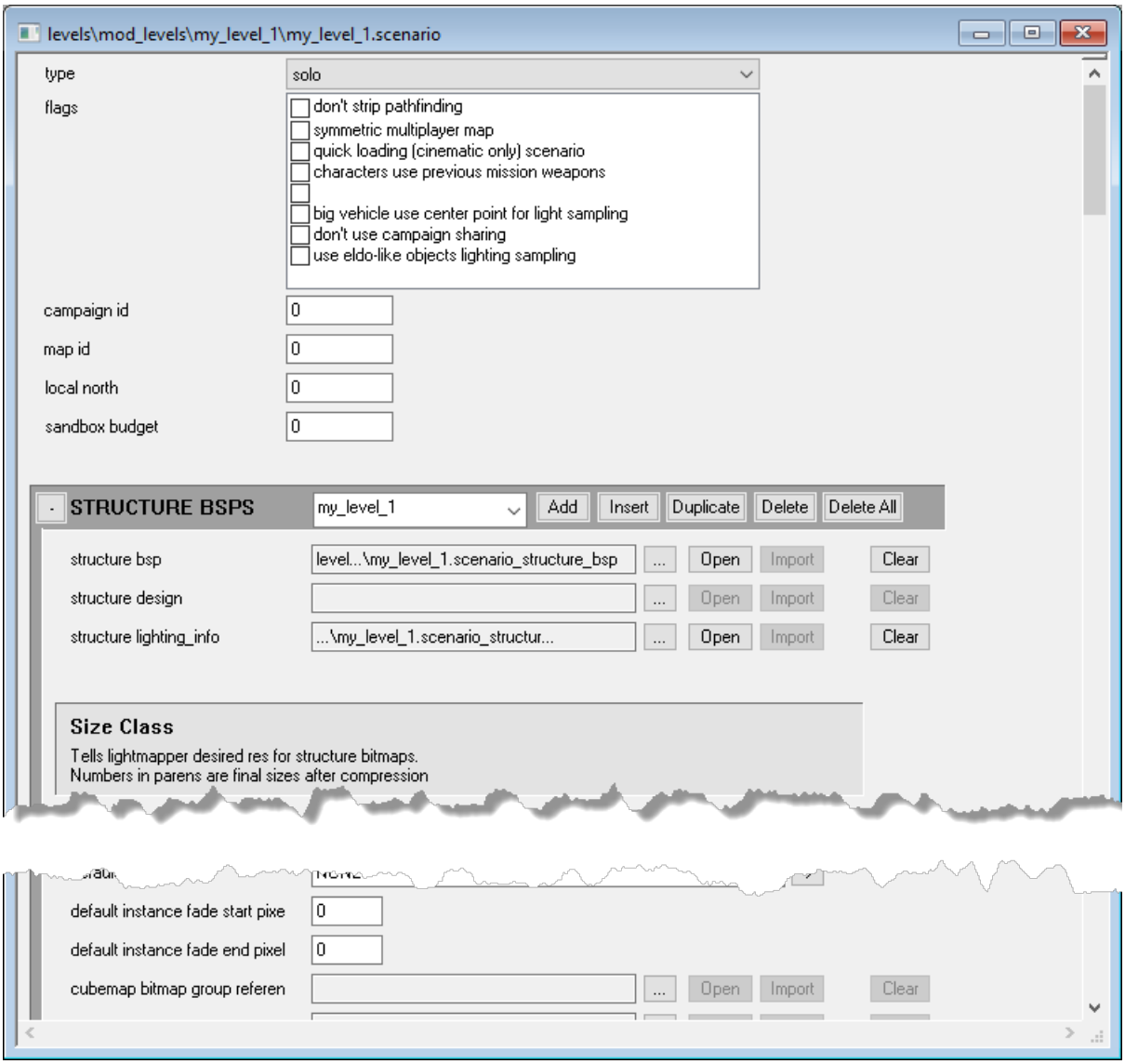


Fig 2. View of the scenario tag within Guerilla.

4. However, there is the **SKIES** section in this dialog.

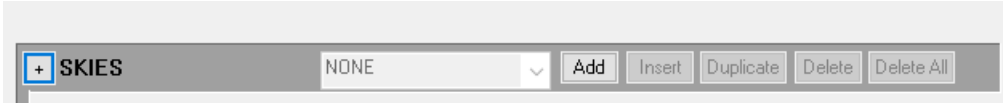
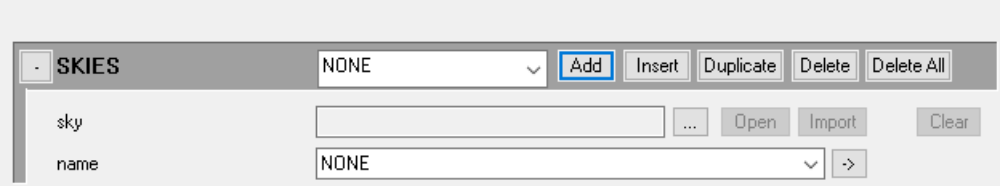


Fig 3. The Skies section in the scenario tag.

5. Click **Add** in the heading of this section. After that, the new sky object will be added and its **name** and **sky** fields will appear.



6. Currently, the name of this sky is **NONE**, which is the default value. Let's keep it so since we just want to add a texture.

7. To add a sky texture (**.scenery** tag of the sky to be more correct):

 - Next to the **sky** field, click the browse button ().
 - In the appearing dialog, proceed to the **tags** folder and find there the **.scenery** tag file with the "**sky**" as part of the name. The easiest way to do it is to search for the **sky** subfolder in the folders of other levels in the **tags** folder. For example, you can select the **...\tags\levels\solo\020_base\sky\sky_01\sky_01.scenery**.

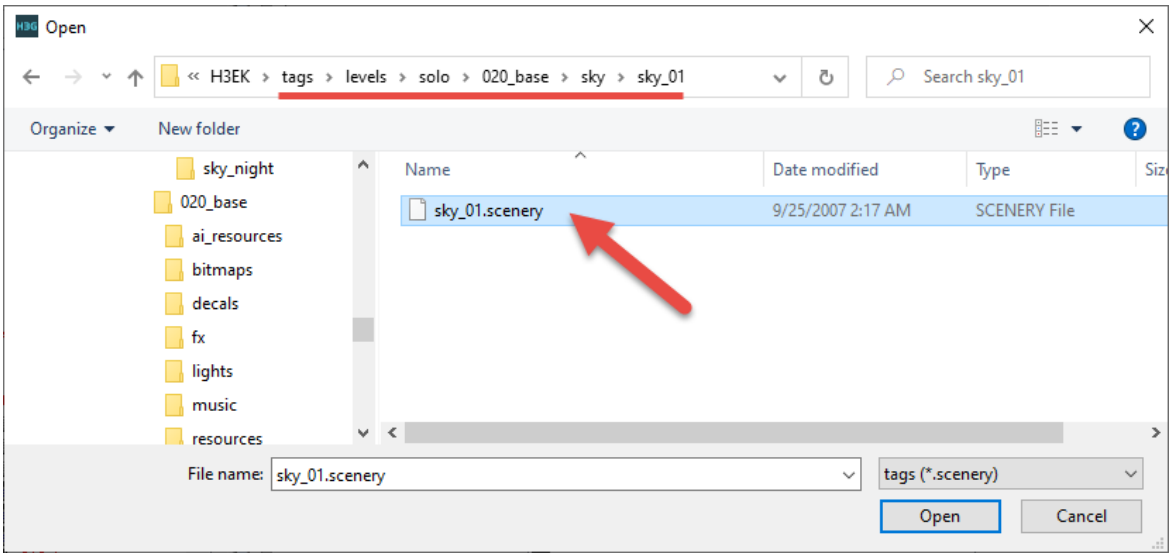
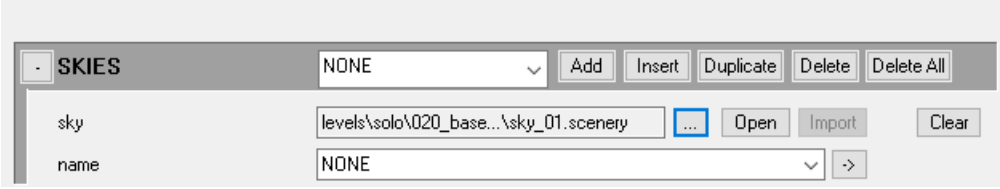


Fig 5. The sky_0101.scenery file in the 020_base level.

 - Now you have assigned a sky to the surfaces of your level with the **+sky** material and your **SKIES** section will look like the following:



8. Save your changes to the .scenario file (File > Save or CTRL + S).

As you can see, the sky is a special case (and its internal mechanics is rather complicated, we will not cover it here).

Assignment of Texture for Material via Creation of Shader Tag

Now, let's assign a texture to the **ground** material. We will do this by creating a **.shader** tag for this material.

Your level does not need any **.shader** tags to be playable, but without them, you will not have any textures. Also, you can have many, many **.shader** tags to specify different shaders for your environment, but for this demonstration we will only be creating one - for the ground.

> **NOTE**
>
> As with skies, we will do it as simply as possible, just to illustrate the process. We will not cover the details that can be necessary for the high-quality levels. E.g., we will not cover the details of creating a shader, or UV maps that need to be created in Blender for surfaces to be appropriately wrapped with textures, or other details.

To create a simple shader with one diffuse texture and link it to your material specified in Blender, you can do the following:

1. In Guerilla, use the file tree to navigate to your folder with the .scenario tag (if you have not already done so).

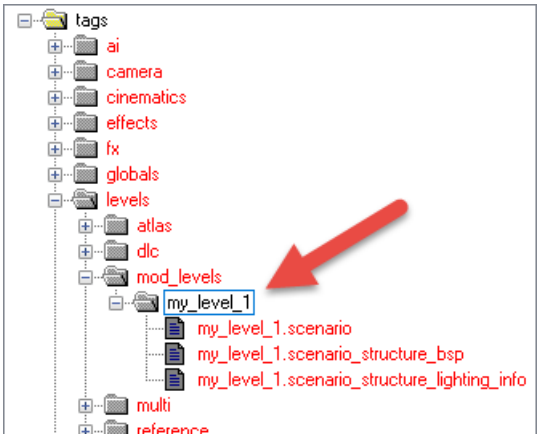


Fig 7. Scenario folder structure.

> **WARNING**
>
> All **.shader** tags of your level need to be saved in a folder called "shaders". To be more precise, all shader tags, except the shared ones.

2. In the main menu of Guerilla, select **File > New**. This will bring up the **New Tag** dialog.

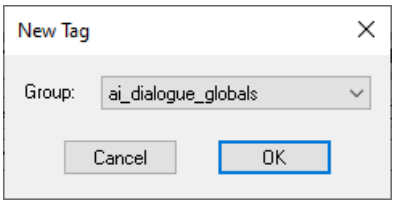


Fig 8. New Tag Dialog.

3. In this dialog, select **shader** as Group (type) and click **OK**. This will create a new **.shader** tag.

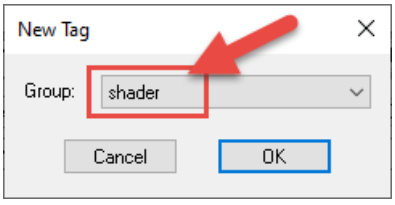


Fig 8. Shader type selected.

4. In the appearing shader tag dialog, scroll down to the **ALBEDO** section.

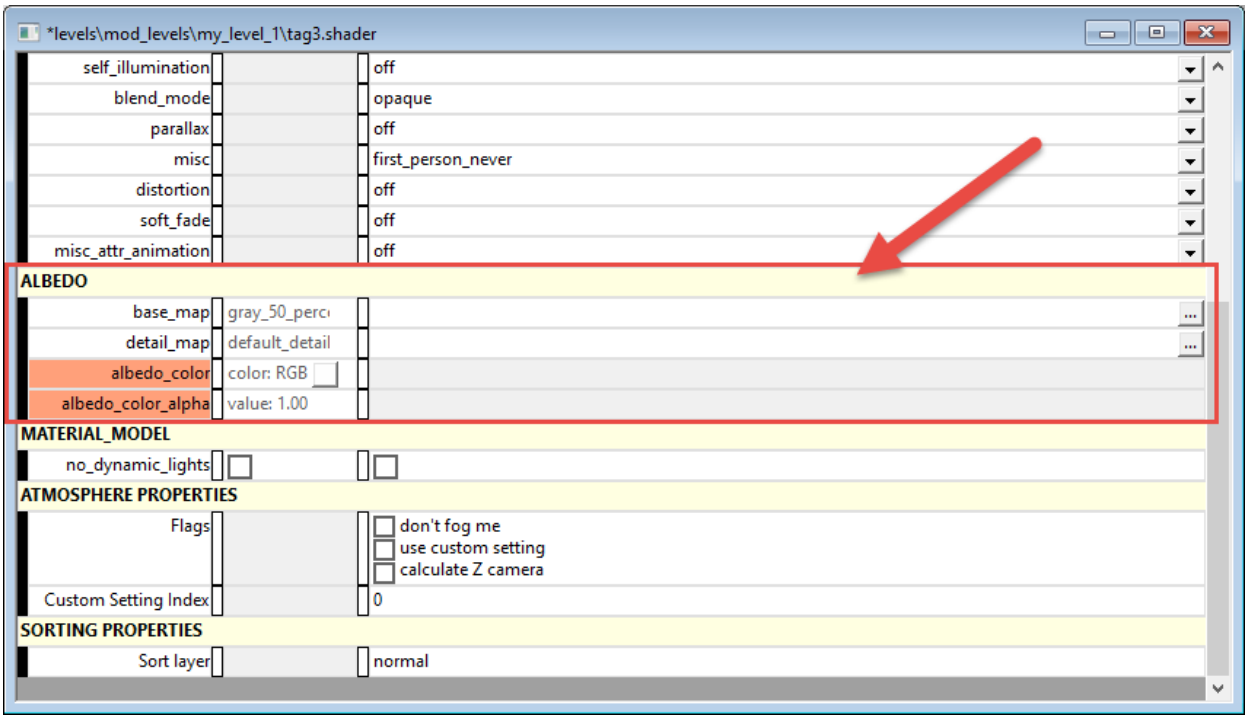


Fig 9. The Albedo section of the shader tag.

5. In the **base_map** field, click the browse button (...) and select the **.bitmap** tag file (texture) to be used as the base texture for this shader. By default, the browse dialog will display default textures of shaders located in **tags\shaders\default_bitmaps\bitmaps**.

 However, let's select another texture for our material, e.g. one of the **.bitmap** tag textures that are provided with the levels.

 For example, let's use the following texture for our **ground** material:
 tags\levels\solo\020_base\bitmaps\hb_ground_dirt_b.bitmap

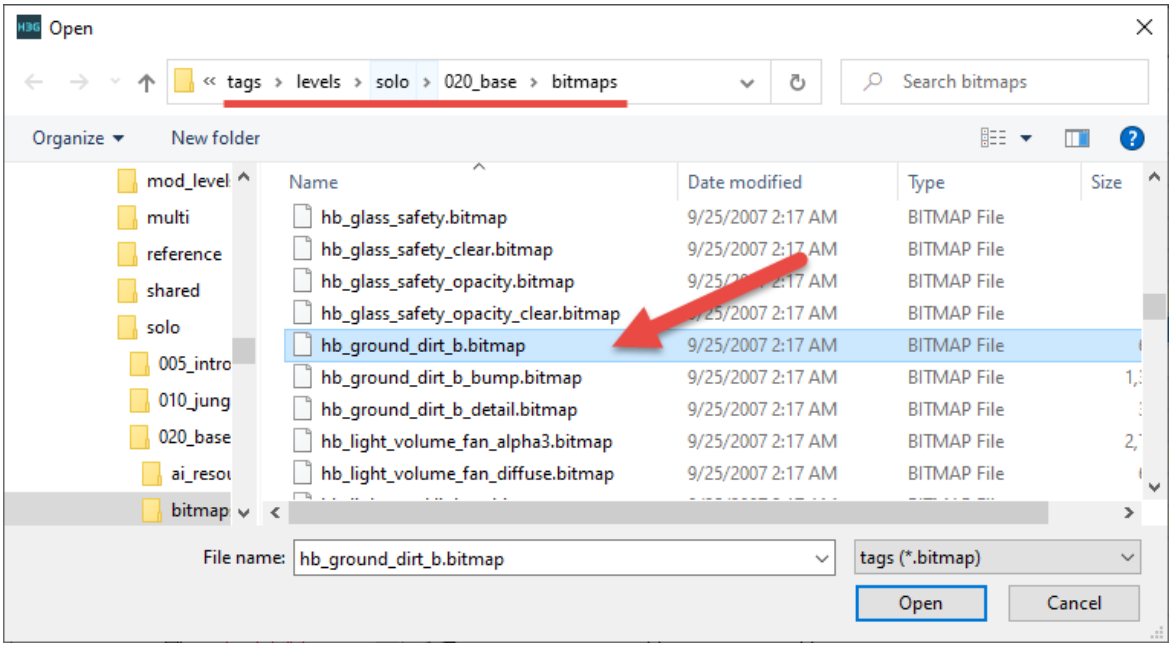


Fig 10. The hb_ground_dirt_b.bitmap to be used for the ground material.

> **NOTE**
>
> Without UV-maps assigned to geometry and models these textures will rarely look very pretty. UV-maps are assigned to geometry and models in Blender and are part of the resulting FBX. However, the process of creating UV maps is not covered by this simple guide.

6. After selection of the shader base texture, you need to save the new **.shader** tag. This is done in a standard way (**File > Save** or CTRL + S).

 However, there are two **very important** nuances here

 - The name of the **.shader** tag needs to be exactly the same as the name of the material that you created in Blender.

 In our case, the material in Blender was called **ground**, so our name of the shader tag will be "**ground.shader**".

> **WARNING**
>
> If you name the shader incorrectly, it will not work for your level in the game engine.

 - You need to create a "**shaders**" subfolder in the folder of your level and save your shader tag file there. This is absolutely necessary since all .shader tags of your level need to be saved in a folder called "shaders". (To be more precise, all shader tags, except the shared ones.) In the **Save As** dialog, you can create a new folder by clicking **New Folder** on the toolbar.

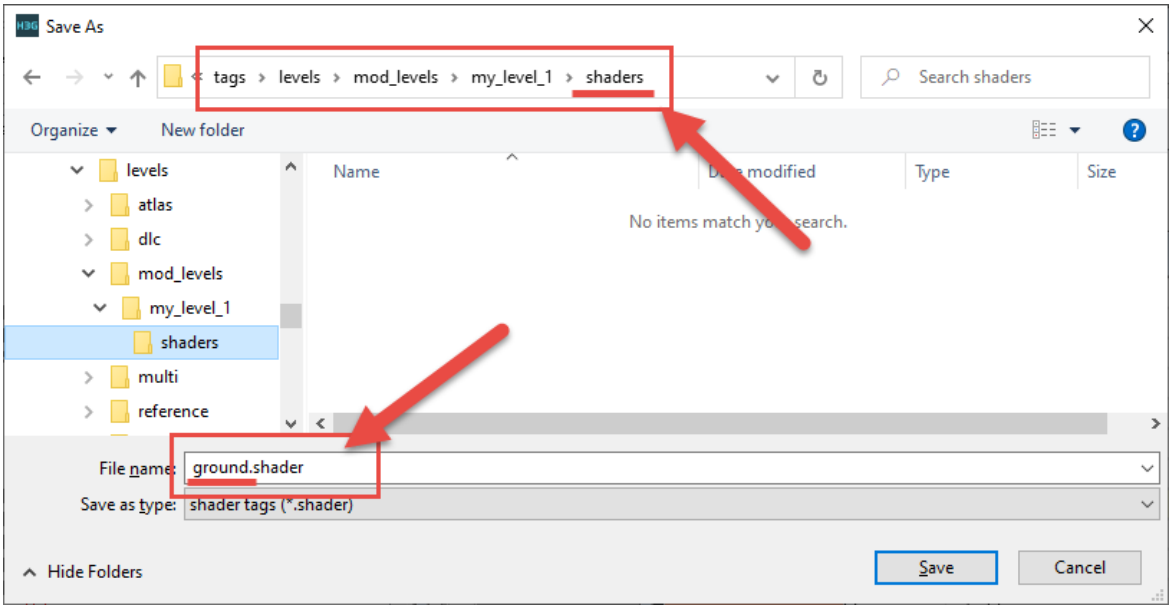


Fig 11. The ground.shader being saved in the shaders folder.

7. After saving your shader tag, close its window in Guerilla. After that, if you press **F5** to refresh the navigation tree and expand the shaders folder, you will see the created file of the shader in the hierarchy.

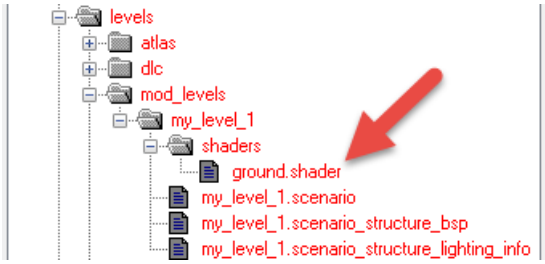


Fig 12. ground.shader file in the Guerilla hierarchy.

After that, you can close Guerilla, since we have selected a sky texture and, via the shader tag, assigned a texture to the Blender material. However, before proceeding to the next step, you need to rebuild your main scenario tag files (**.scenario** and others) to link the level to the assigned shader texture.

Rebuilding Main Scenario Tag Files

To rebuild the main scenario tag files from the ASS file, you will need to execute again the same **structure** command of the **tool_fast.exe**.

For example, in our case:

```
tool_fast.exe structure levels\mod_levels\my_level_1\structure\my_level_1.ass
```

For details, see Step #4.

After regenerating these tag files, you can proceed to the Calculation of Lighting step.

> **NOTE**
> Moreover, if you are eager to view the geometry of your level with textures (but still without lighting), you can simply open the generated **.scenario** tag file (e.g. **my_level_1.scenario**) using Sapien.

Quick Start Process Step 6 - Calculation of Lighting

11/23/2022 • 3 minutes to read

The lighting of your level needs to be calculated using the **calc_lm_farm_local.py** Python script, which is located at the root directory of the Halo 3 Editing Kit. To launch this script, you will need to install Python.

> **NOTE**
> Versions greater than **Python 3.6** should be used for launching this script.

To install python:

1. Open https://www.python.org/downloads/windows/
2. In the **Stable Releases** section, using notes on that page as a source of info, select a version of Python that works on your version of Windows (for example, Python **3.10.0 will not work on Windows 7 or earlier**). For example, we will choose to install **Python 3.10.0**, which is appropriate for **Windows 10**.
3. By clicking the corresponding link, download the **Windows installer** of this version. It's not very important whether you select the 32-bit or 64-bit version of Python.



Fig 1. Download the Windows Installer for Python.

4. Launch the downloaded installer.

5. While installing, we recommend you to enable the **Add Python ... to PATH** option, which will add the path to your python executable to the PATH environment variable. This will allow you to execute python scripts by simply typing **python <name_of_script.py>** from any directory.

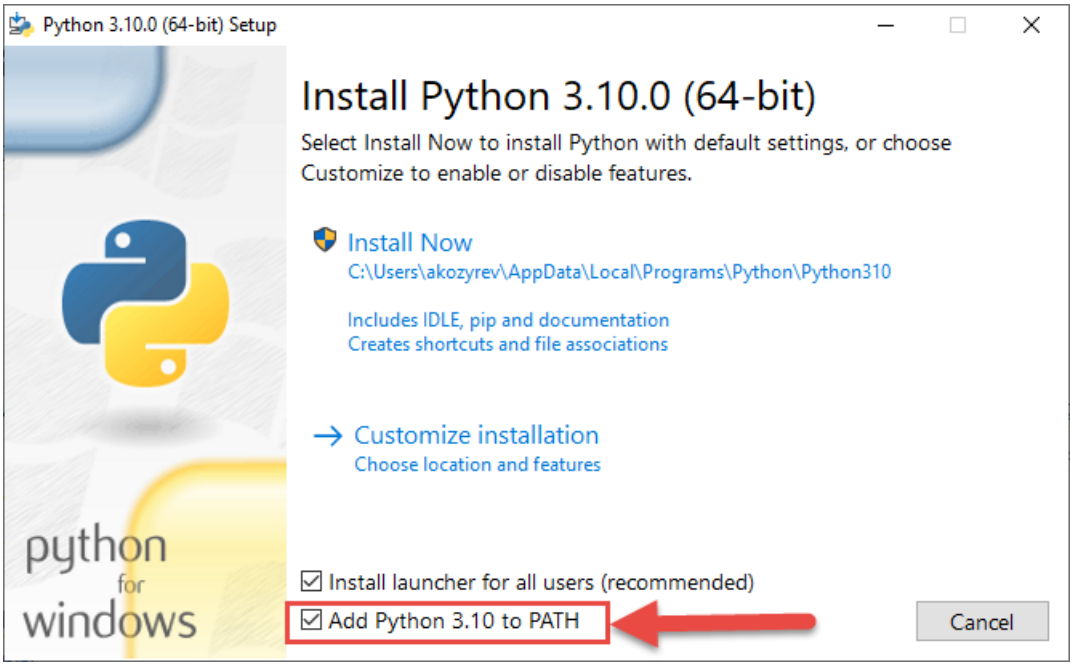


Fig 2. Install Python and make sure Add Python 3.10 to PATH is checked.

6. Select either the default install (**Install Now**) or customize your python installation, if you like.

7. After installation, open a new command prompt window (see Step #3 for details) and execute the **python** command without the parameters. If the installation has been successful, Python will display its version and start a Python Interpreter in the same command prompt window.

 To exit the Interpreter and return to the original command prompt, type **quit()** and press ENTER.

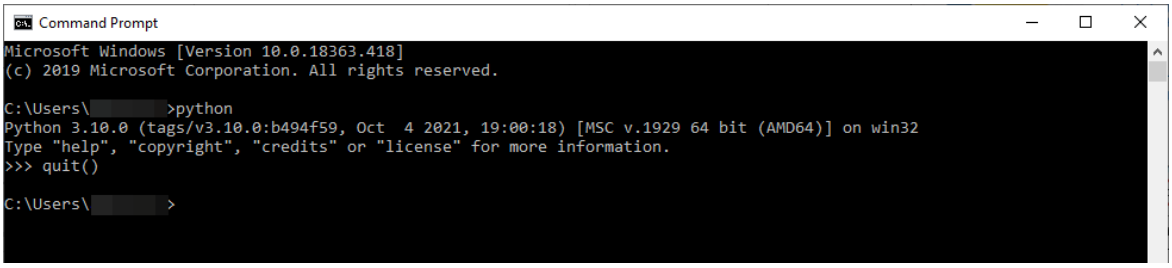


 Fig 3. The Python Intepreter command prompt.

8. Now you can execute python scripts from the command line.

After Python is installed, you will be able to launch the **calc_lm_farm_local.py** script and calculate lighting by executing a single command in the command prompt.

> **NOTE**
> Typically, to be able to omit the path to the **calc_lm_farm_local.py** script itself, this command is executed from the root directory.

The format of the command is the following:

```
python calc_lm_farm_local.py <scenario tag file, w/o .ext> <bsp name> <quality> (<light_group>)
```

Where the parameters are the following:

- **<scenario tag file, w/o .ext>** – The path to the **.scenario** tag file in this command is specified relative to the **tags** folder. The name of the **.scenario** tag file is specified without the file extension. E.g., in our case, **levels\mod_levels\my_level_1\my_level_1**
- **<bsp name>** – The name of the BSP (Binary Space Partition) for which the light is calculated. In our case, we have a simple level and this name equals to the name of its ASS file. E.g. **my_level_1**

> **NOTE**
> A level can be divided into multiple BSPs (Binary Space Partitions) during its creation in the 3D modelling program. However, most of the custom levels have a single BSP only. Large campaign levels may have multiple BSPs. The BSP concept is out of the scope of this tutorial.

- **<quality>** – the quality of the calculated lighting. Please note that the higher the quality is, the more time the calculations do require. The possible values here are:
 - **high**
 - **medium**
 - **low**
 - **direct_only**
 - **super_slow**
 - **draft**
 - **debug**
- **<light_group>** – optional parameter, can be omitted. This is the name of the light group, if it is specified, the light will be calculated not for the whole level but for its part. In our tutorial, we will omit this parameter.

For example, in our case, the command can be the following:

```
python calc_lm_farm_local.py levels\mod_levels\my_level_1\my_level_1 my_level_1 low
```

This command will be executed in the command prompt for some time, you will see a lot of debug info displayed in the command prompt at this time (errors, warnings, etc.). Please wait until the program finishes.

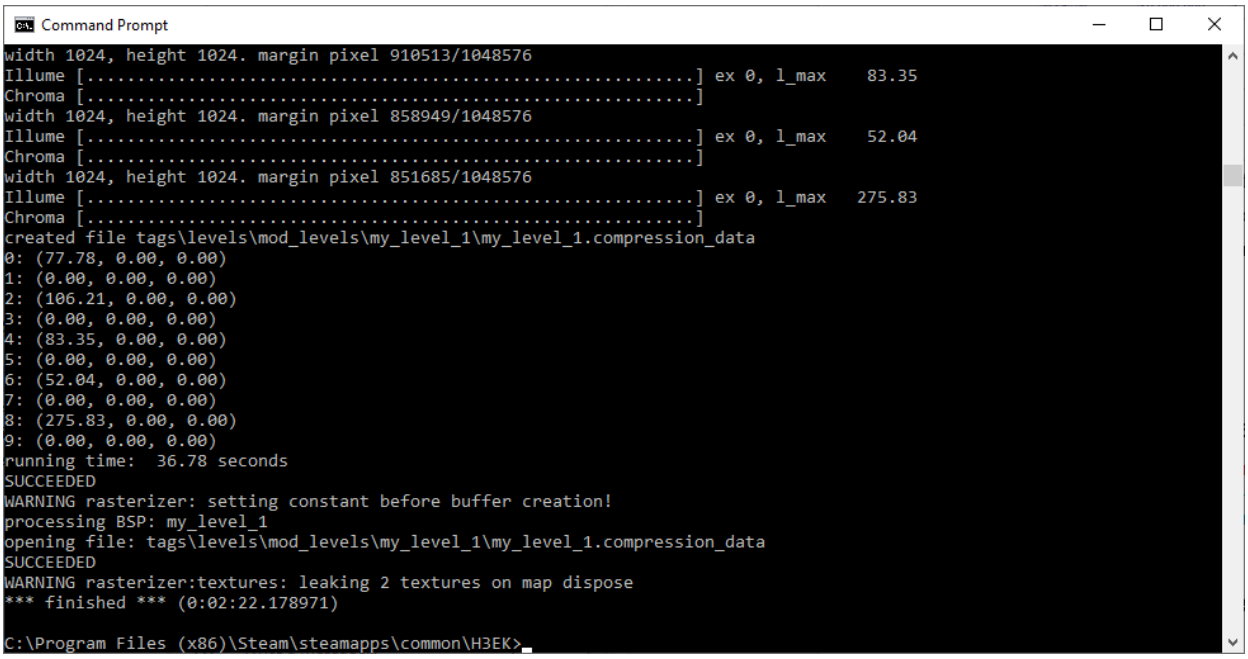


Fig. 4. Execution of the script.

The execution of this command will result in the calculation of lightmaps and other light data. As a result, a lot of tag files related to lighting will appear in the folder of your level in the **tags** folder:

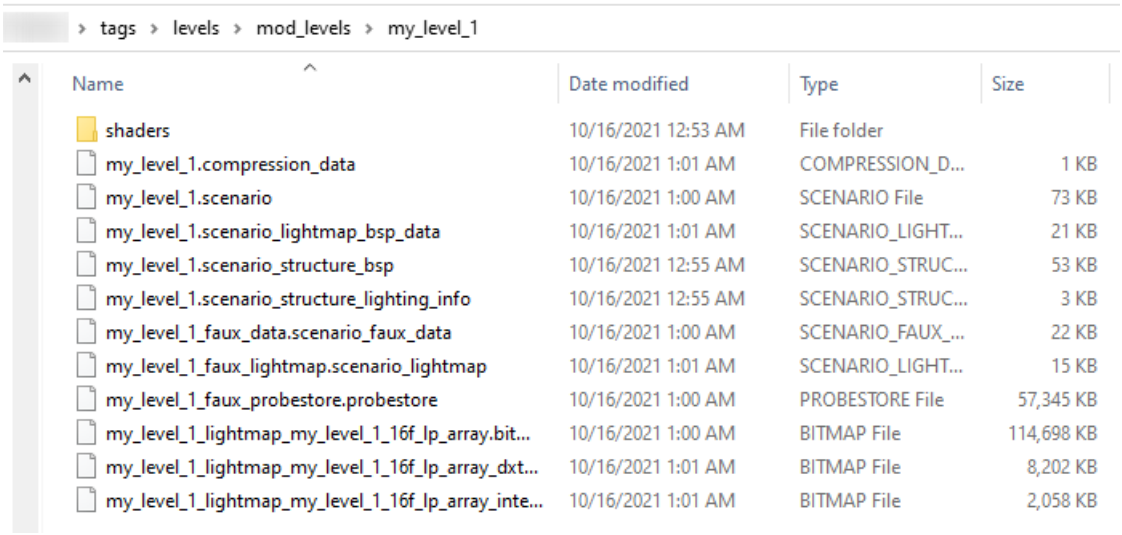


Fig 5. Lighting files output.

After that, you can proceed to the Populating the Level in Sapien step.

Quick Start Process Step 7 - Populating the Level in Sapien

11/23/2022 • 8 minutes to read

Now, you can preview your level in Sapien (with textures and calculated lighting), add spawn points to it, and, if you like, add some other objects to it. See the subsections below for details.

Previewing Your Level

To preview your level, you need to open its **.scenario** tag file (see Step #4) in Sapien.

To do this, launch **sapien.exe** (located at the root directory of Halo 3 Editing Kit) and, in the **Open** dialog, proceed to the folder of your level in **tags** (e.g. **...tags\levels\mod_levels\my_level_1**) and open your **.scenario** file.

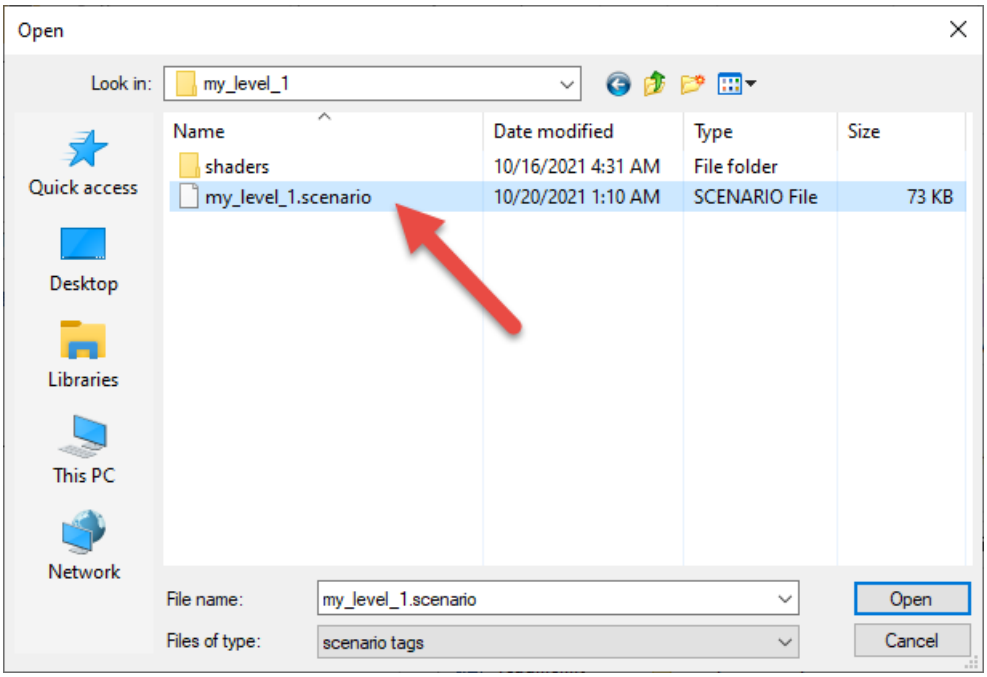


Fig 1. Open your .scenario in Sapien.

After that, your will see your level loading in the **Game Window** of Sapien, which will display a lot of debug info in the process. The loading of the level can take some time.

After your level is loaded, you will not see it. All that you see at the first glance will be the sky and the horizon (see below).

You will need to move your camera to your level to see it clearly. The controls for moving the camera in the **Game Window** are the following:

- **Pressing and holding the mouse wheel + moving the mouse** – rotates the camera.
- **W + holding the mouse wheel** – moves the camera forward.
- **S + holding the mouse wheel** – moves the camera backward.
- **A + holding the mouse wheel** – strafes the camera left.
- **D + holding the mouse wheel** – strafes the camera right.

- **R + holding the mouse wheel** – strafes the camera up.
- **F + holding the mouse wheel** – strafes the camera down.
- **SHIFT + holding the mouse wheel** – will toggle your movement speed.
- **CTRL + [W or S or A or D] + holding the mouse wheel** – will move the camera with the increased speed (only while you holding CTRL + mouse wheel).

Using these controls you will be able to locate your level. Particularly, since we have created our level in the origin of coordinates (relative to the location of the frame node), you will need to move your camera up a little bit and rotate it.

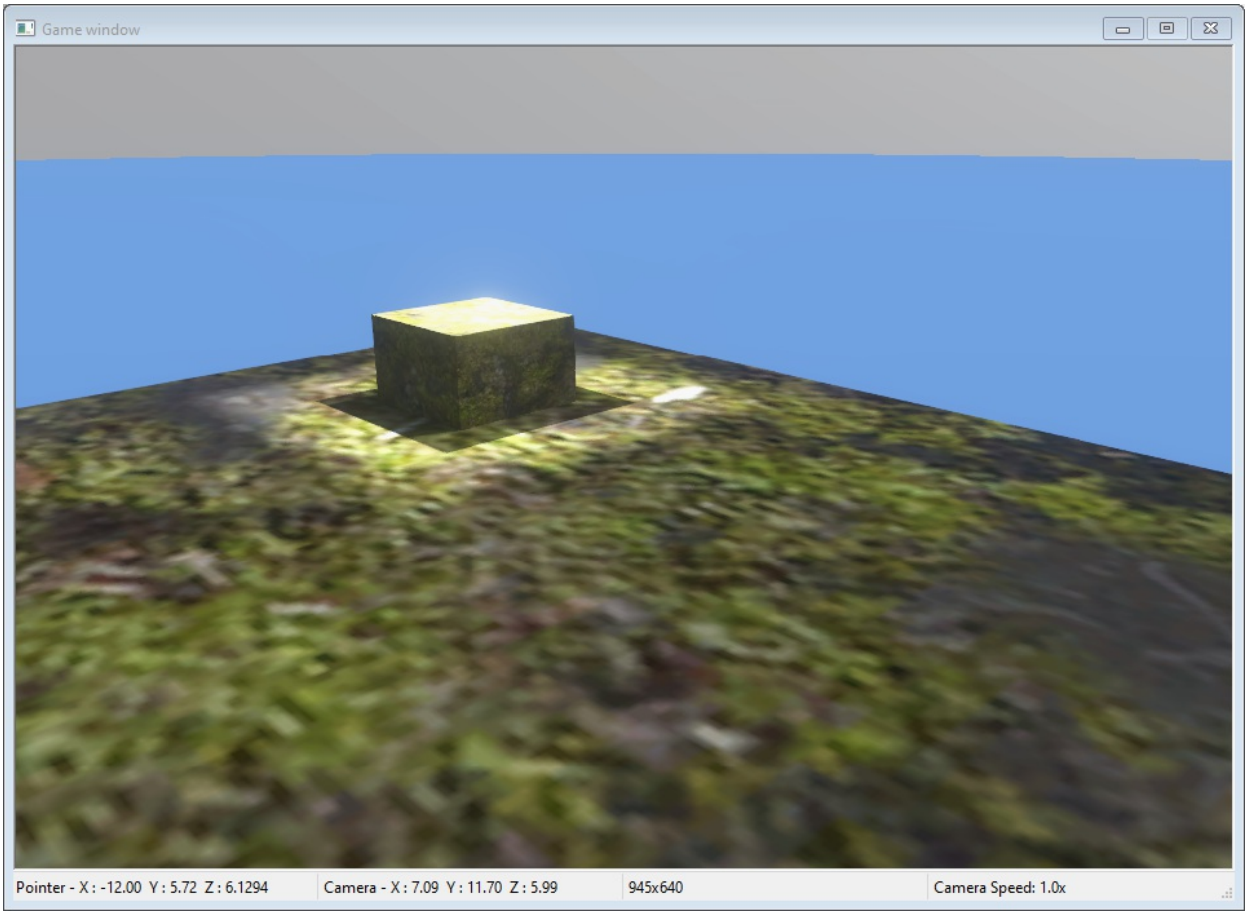


Fig 2. Your created level in Sapien.

Finally, you will see that the level looks as intended: there are no walls of the large cube (since we have assigned the **+sky** material to them), the floor has the assigned ground texture and there is a small cube with the light spot above and the shadows. And, if you look up, you will see a nice sky.

Now, you can press **TAB** to try to spawn a Master Chief on your level. As a result, you will get an error, since your level has no spawn points. So, it's time to add them.

Adding Spawn Points

To add a spawn point:

1. In Sapien, go to the **Hierarchy View** window and click on the + symbol next to the **Scenario** folder to expand it.
2. Click on the + next to **Game Data** to expand that as well.
3. In the **Game Data** list in the Hierarchy View window, select **Player Starting Points** by clicking on it. This is the mechanism we will use to add a spawn location for the player.

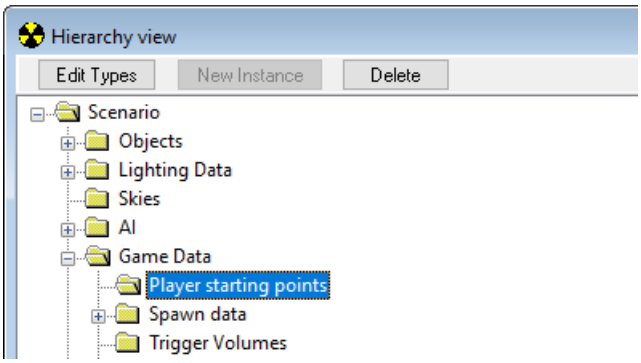


Fig 3. Player Starting Points folder.

4. Go to the **Game Window** and ***right-click*** to place a spawn location. You should see a **Manipulation Gizmo** appear for the Spawn Point (see below).

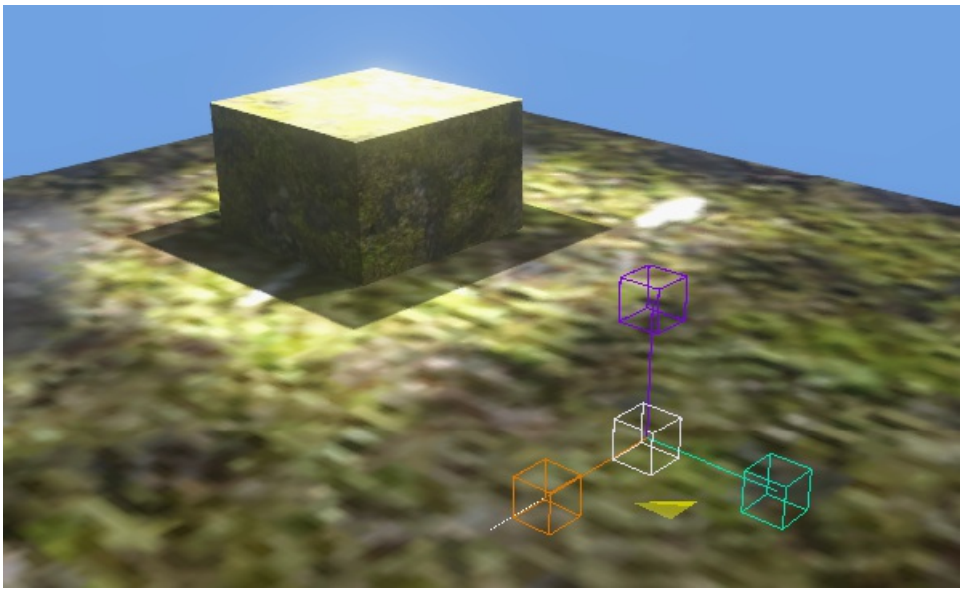


Fig 4. The object Manipulation Gizmo.

5. Also, look at the **Hierarchy View** window. Notice that a new item has appeared under the **Element** heading. This is the unique identifier for our spawn point. You can click on it at any time to select the spawn point.

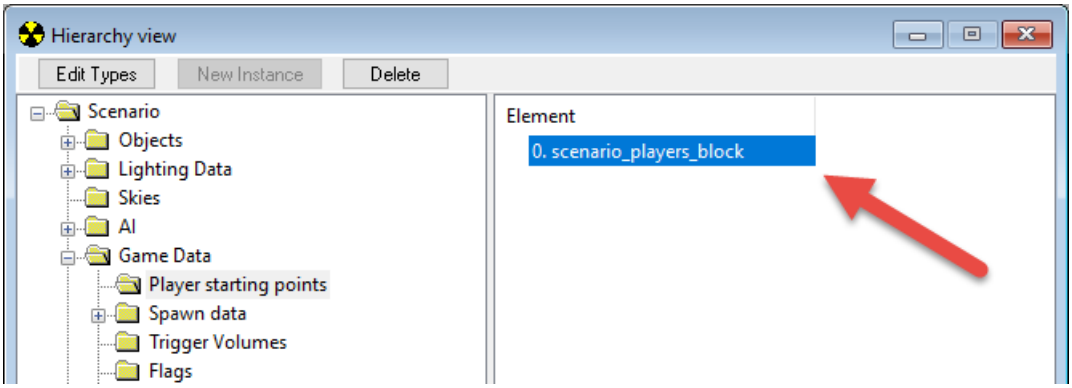


Fig 5. The newly placed object in the Sapien hierarchy view.

> **NOTE**
>
> If you have an object with its type already selected in the **Hierarchy View** window and you right-click on the **Game Window**, a new instance of that object type will be created and selected.

6. We've got our first spawn location, but it may not be on the ground or in the exact location that we want

it. This is where the **Properties Palette** comes in. Make sure the spawn point is still selected and then switch to the **Properties Palette**.

7. You should see an item labeled **Position**. For now, we don't care about the **X** and **Y** values, but change the **Z** value to 0 – this will ensure that the player starts out on the ground.

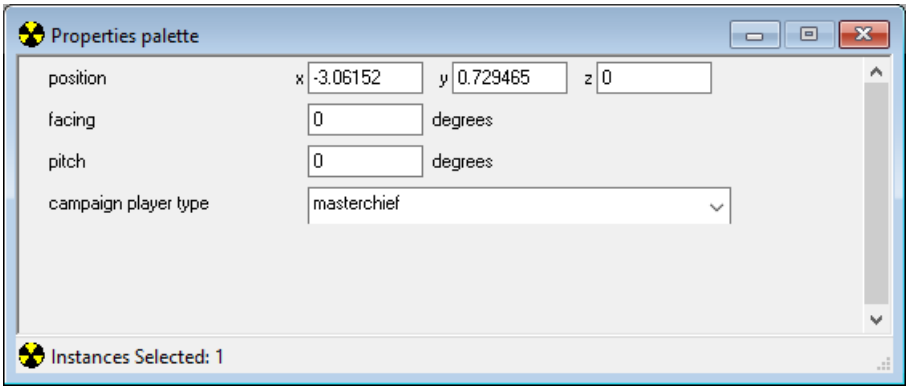


Fig 6. Set the z axis to 0 for the position.

8. To save your scenario, in the main menu select **File > Save** (or press CTRL + S).

After that, you can spawn Master Chief by pressing **TAB** in the **Game Window**. Pressing **TAB** again will switch you back to the free camera mode.

We could start the map like this, but what's a FPS without a weapon? Let's give our player a weapon to start with.

Adding a Starting Weapon via Starting Profile

To add a starting weapon:

1. In the **Hierarchy View** under **Game Data** is an item called **Starting Profiles**. Click on it to select it.
2. At the top of the **Hierarchy View** is a button labeled **New Instance**. Click it. This will create a new instance of a starting profile for us to edit.

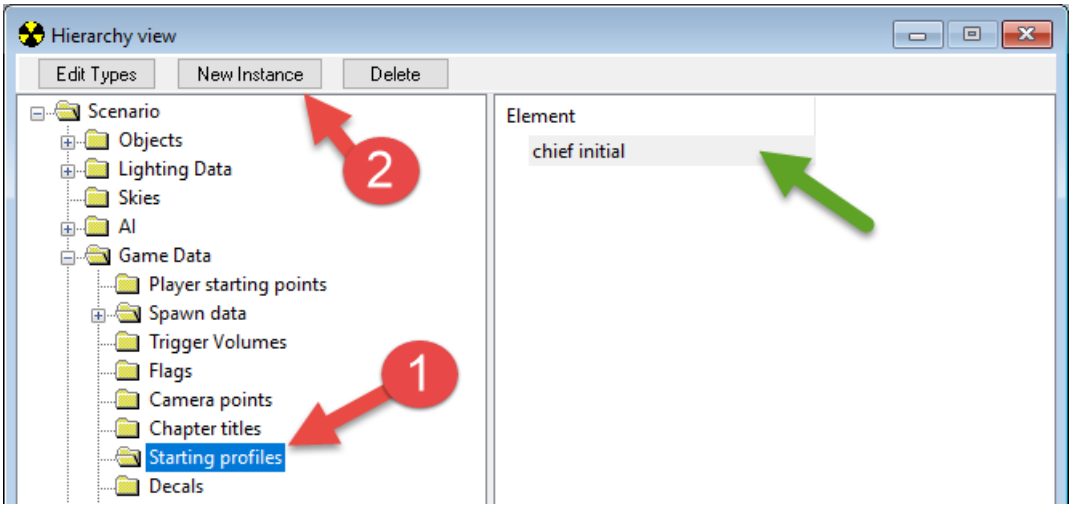


Fig 7. Create a New Instance in the Starting Profiles folder.

3. Now switch to the **Properties Palette**. You can change any of the values for the starting profile (including the name).

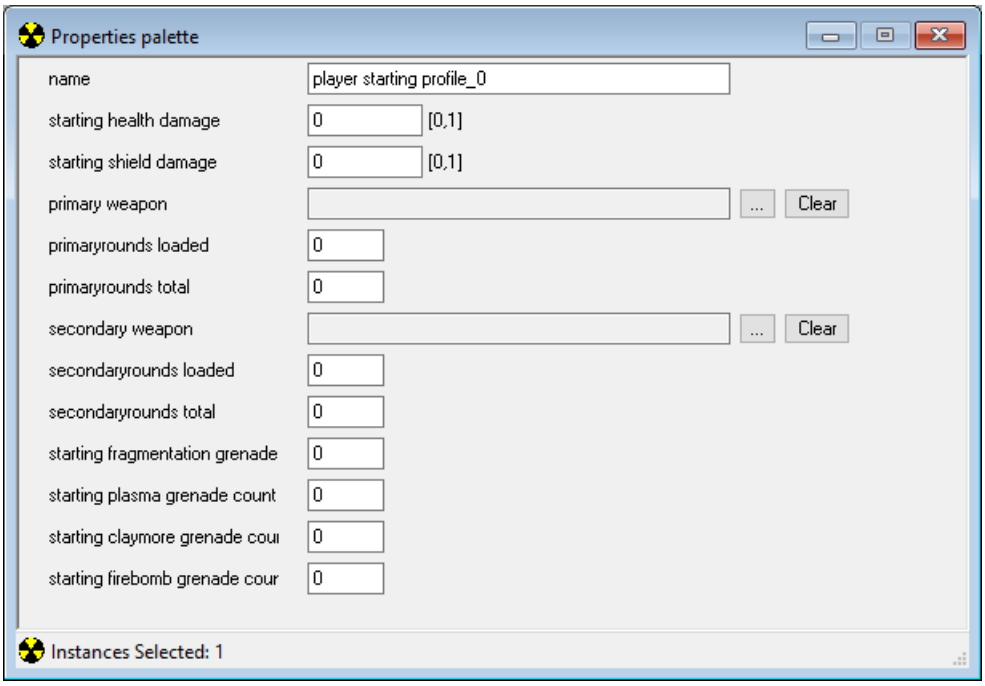


Fig 8. Starting Profile in the Properties Palette.

4. Click on the browse button (...) next to the **primary weapon** field. This will bring up a file browser dialog. Navigate to **tags\objects\weapons\rifle\battle_rifle** and select **battle_rifle.weapon**. Click **Open**. This assigns our starting profile a battle rifle for its primary weapon.

5. Be sure to place numbers in the **primaryrounds loaded** and **primaryrounds total** categories as well, or the weapon will not appear in-game.

6. To save your scenario, in the main menu select **File > Save** (or press CTRL + S).

However, if you have already spawned Master Chief before adding the starting profile using TAB, you will still be without a weapon and will probably wonder how to respawn with your modifications applied.

One of the ways to do it is to select **Scenarios > Map Reset** from the main menu (or press ALT + R) to reset the map and then press **TAB** again.

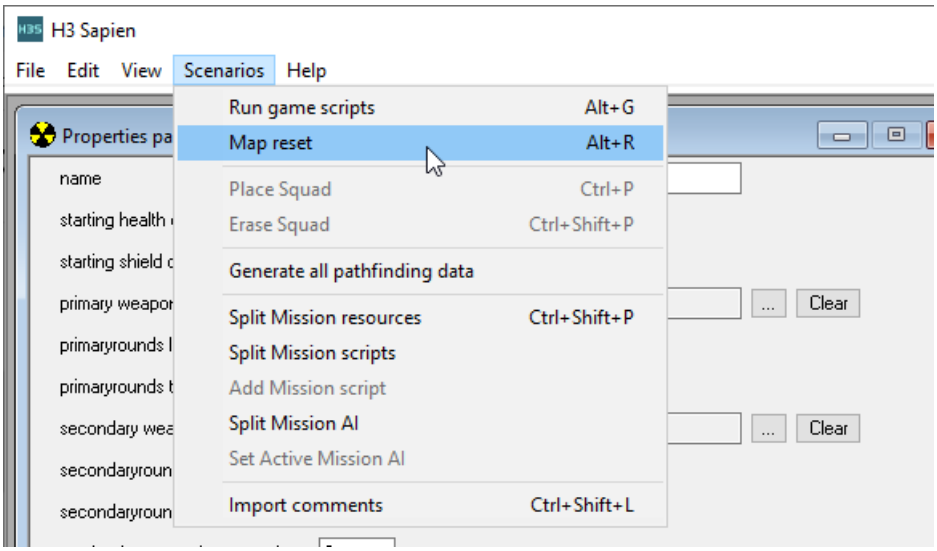


Fig 9. Map Reset option.

Yeah, the good old battle rifle is now in place.



Fig 10. Now Spawning with the Battle Rifle.

Adding an Object (Vehicle)

However, a vehicle would also be good for our level, would not it? Let's add it too. For example, let's add a Warthog.

> **NOTE**
>
> If you have spawned Master Chief on your map using **TAB**, then, before adding the Warthog, switch back again to the free camera view by clicking **TAB** once again.

To add a Warthog:

1. In the **Hierarchy View** window, expand the **Scenario** folder, then the **Objects** folder. Click on the category of object (in our case **Units** > **Vehicles**) you want to add to highlight it.

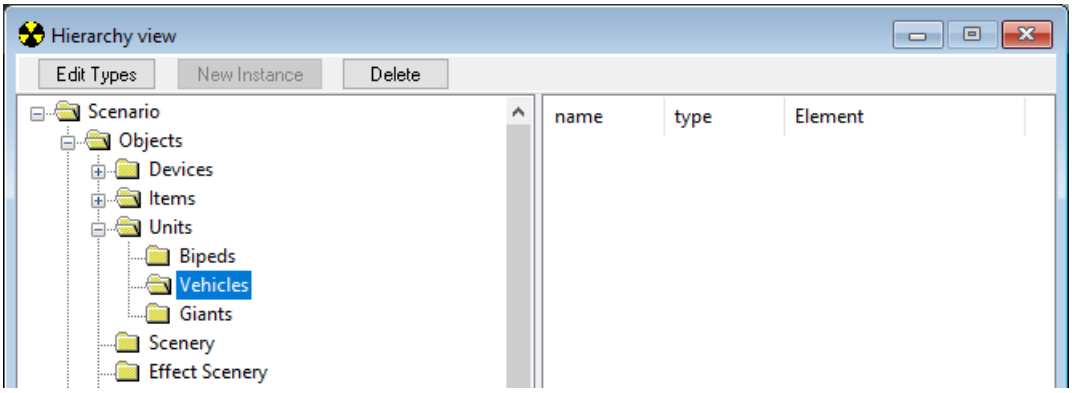


Fig 11. The Vehicles folder in the Hierarchy View.

2. In the **Hierarchy View** window, click the **Edit Types** button.
3. In the appearing dialog (asset type palette), from the **Object class** drop-down list, select the type of object you want to add (in our case **vehicle**). Click the **Add** button.

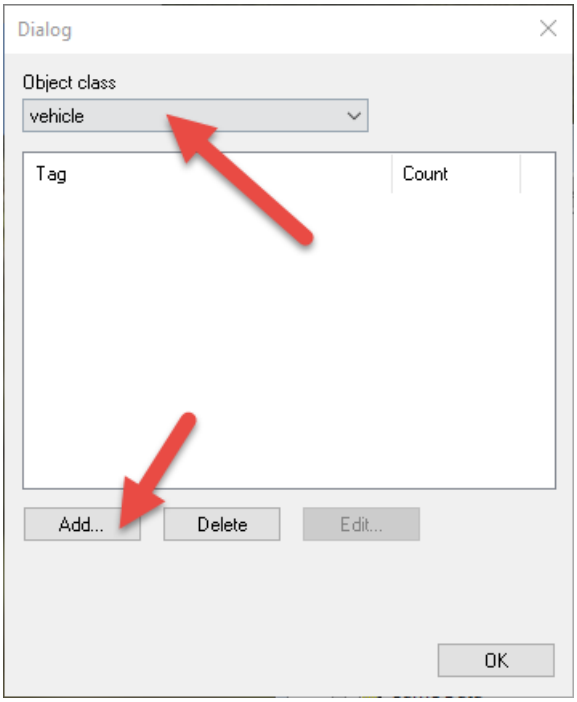


Fig 12. Set the Object Class to vehicle and press Add...

4. In the appearing dialog, find the tag file for the appropriate object. In our case, select the **tags\objects\vehicles\warthog\warthog.vehicle** file tag. Then, click **Add Tags**. You will see that this tag was added to the list of Tags at the bottom of the window.

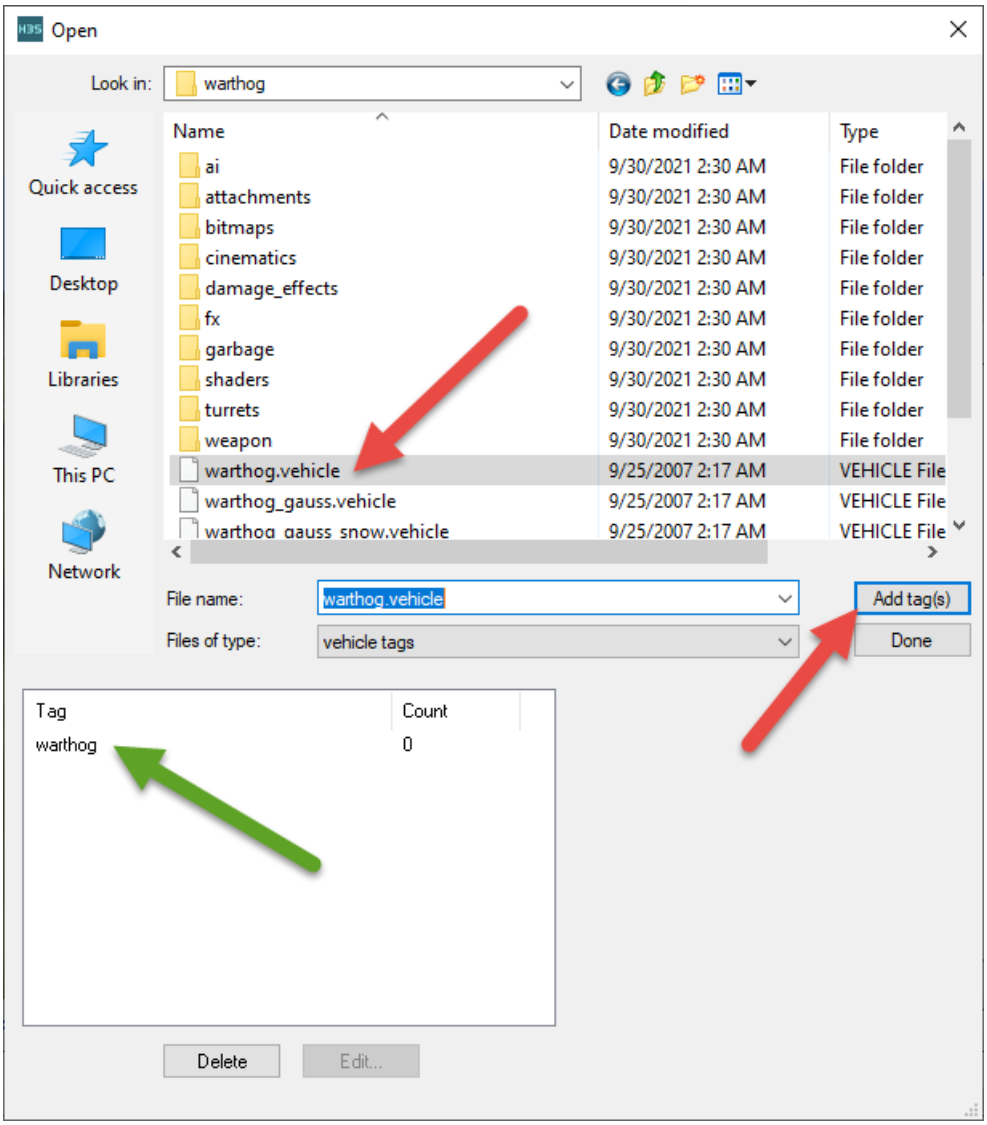


Fig 13. Add warthog.vehicle tag to the map.

5. Click **Done** to close the selection window. You will see that the **warthog** is listed in the asset type palette now. Then, click **OK** to close the asset type palette.

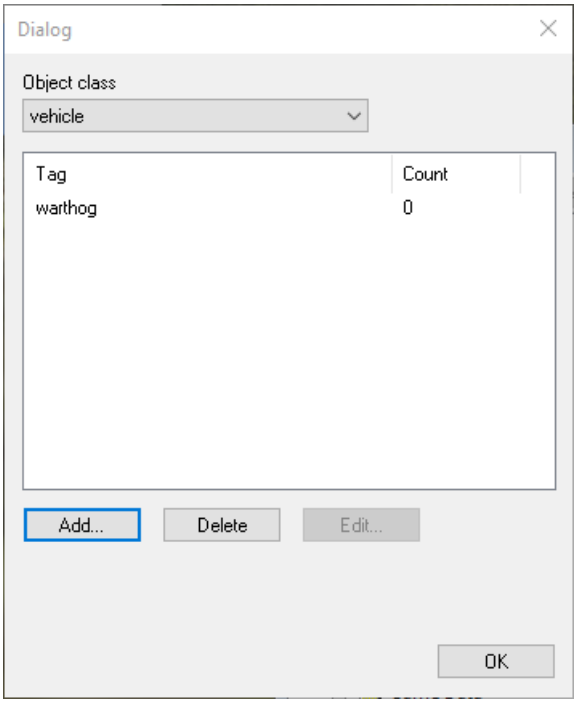


Fig 14. The warthog added to the vehicle object types.

6. Now that you've added the object (in our case, of the **warthog** type) to your scenario, it's time to place an instance of it. In the **Hierarchy View** window, click the category of object you just added to highlight it.

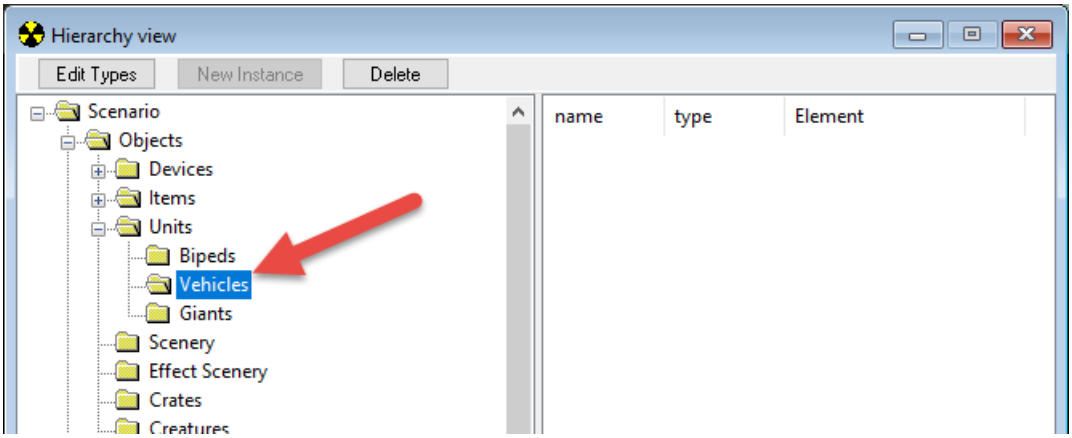


Fig 15. The vehicle folder in the Hierarchy View.

7. In the **Game Window**, right-click to place an object. You'll notice that when the object is placed, an entry appears in the **Hierarchy View** (with Type: None), but in the Game Window only the manipulate gizmo appears. This is because you still need to tell Sapien what the type of the object is you just placed.

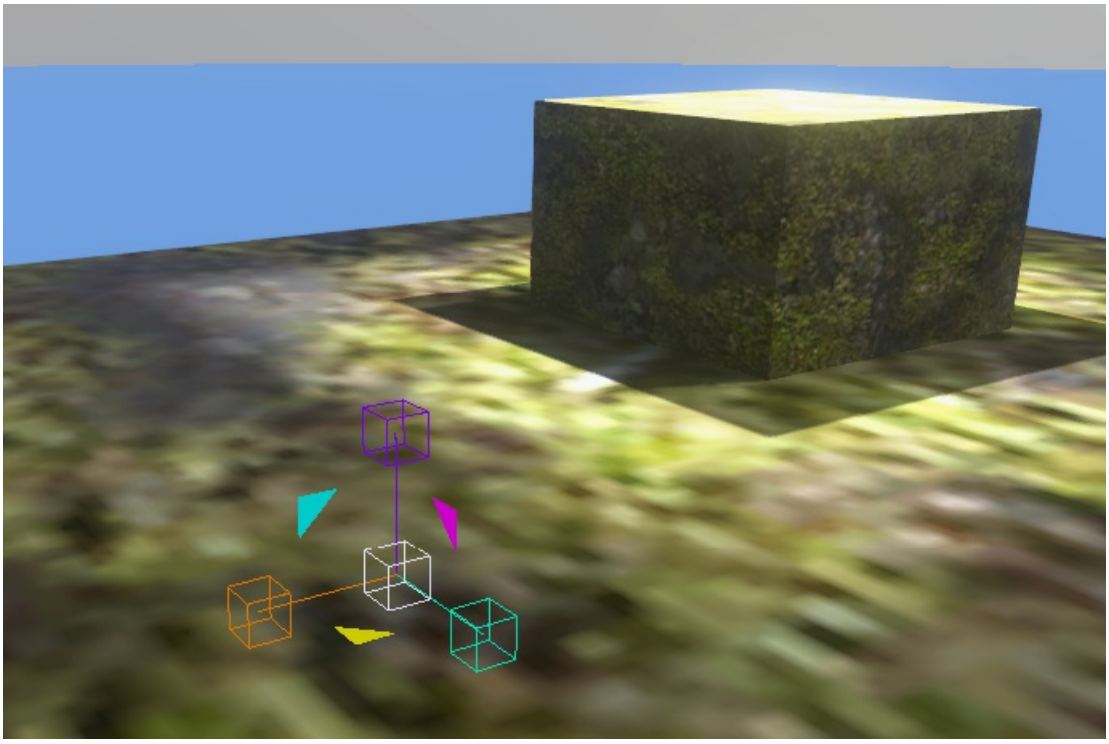


Fig 16. Object with no type in Game Window.

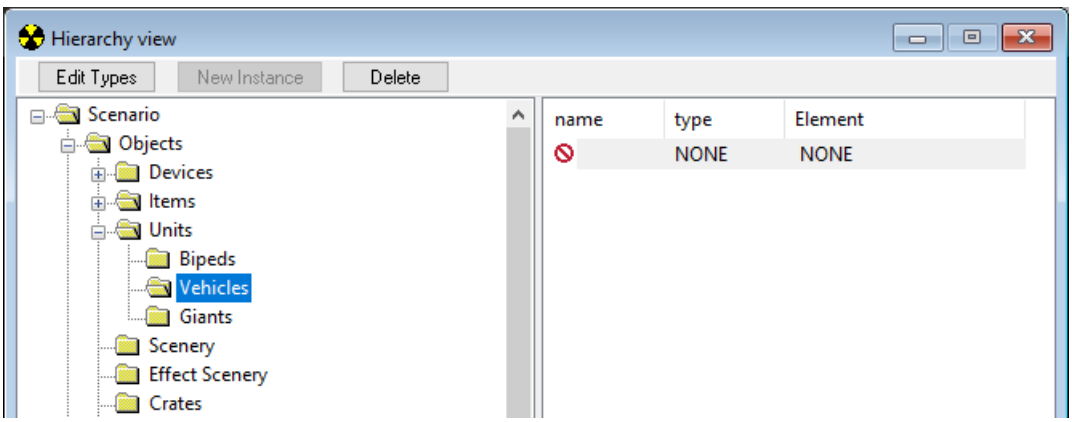


Fig 17. Object with no type in Hierarchy View.

8. With your object still selected (you can click on its entry in the **Hierarchy View** window to be sure), switch to the **Properties Palette** window. In the **Properties Palette** window, click on the **Type** drop-down menu and select the object you added to your palette. If your object doesn't appear in this drop-down, you most likely added an object from the wrong category in the **Hierarchy View**.

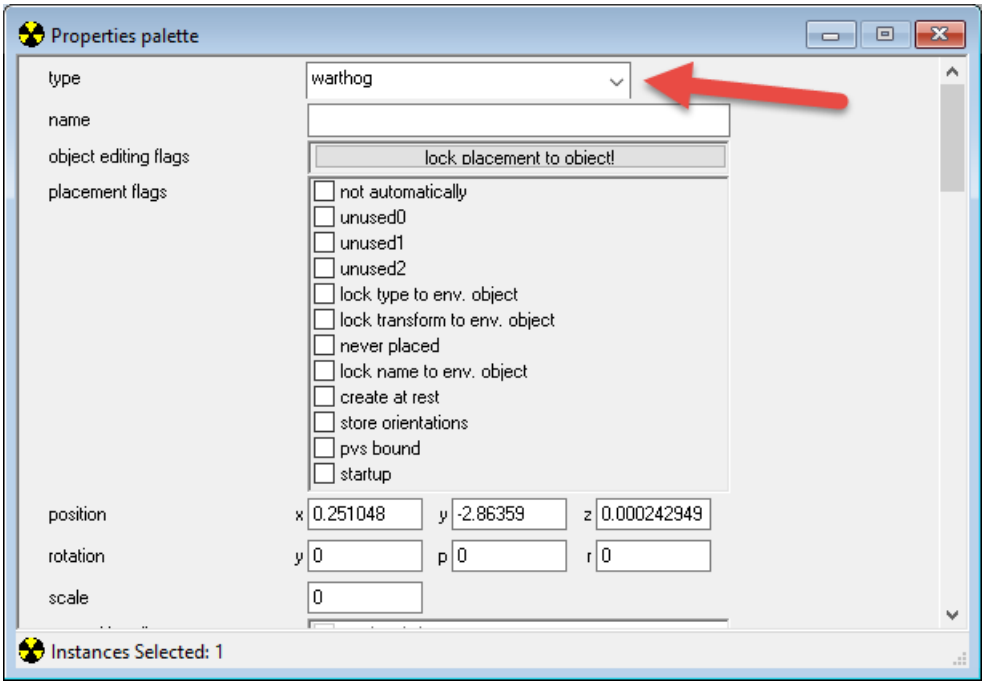


Fig 18. Changing the object type to warthog.

9. Once you've selected a type, you should see your object appear in the **Game Window**.

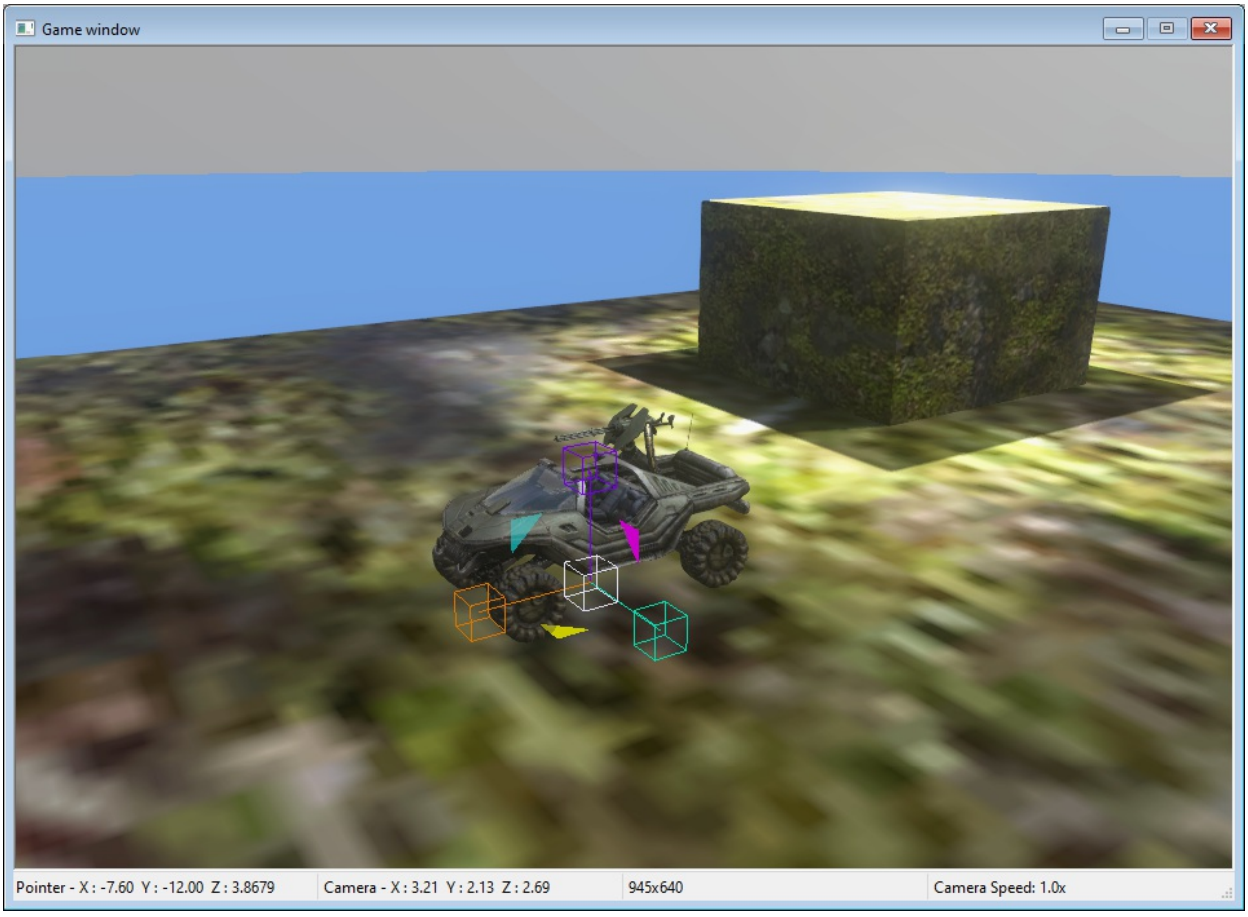


Fig 19. Warthog is now in your level.

10. If necessary, modify other properties of the object in the **Properties Palette** window (e.g. **z** coordinate

in **position**, as before for the spawn points).

11. To save your scenario, in the main menu select **File > Save** (or press CTRL + S).

> **TIP**
> The procedure described above for the Hog is typical for adding other types of objects. You can use it to add weapons, equipment, and so on.

Now, you can test the added vehicle by spawning again as Master Chief (**TAB**) and using it (hold **E**, just as in the game).

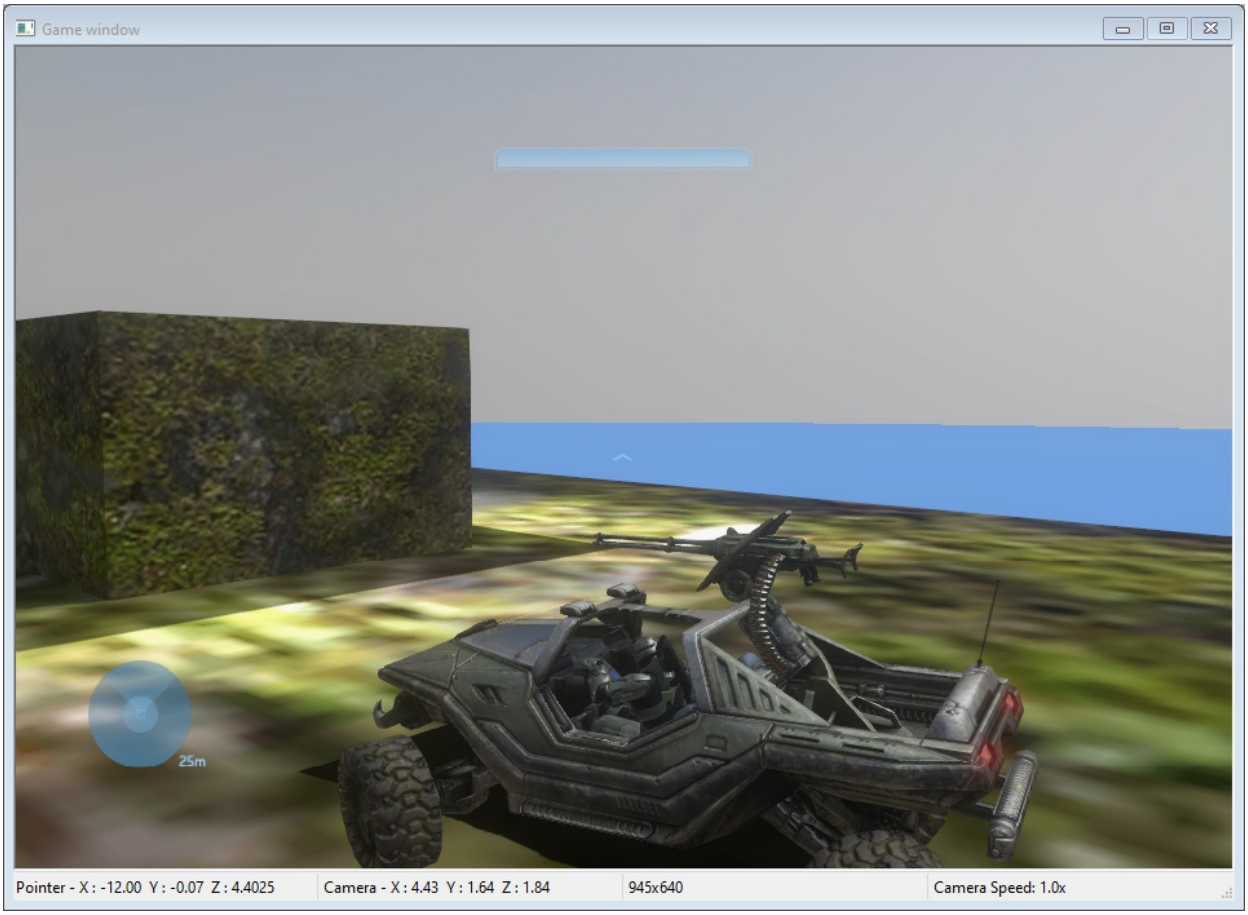


Fig 20. Warthog playable in your level.

After that, the level seems to be ready and you can proceed to the Loading the Level from Tags step to test it.

Quick Start Process Step 8 - Loading the Level in Sapien

11/23/2022 • 2 minutes to read

Now, the level is ready and you can start testing it.

As one of the options, you can load it with the help of the **halo3_tag_test.exe**, which is a specifically modified version of the Halo 3 game that allows you to fully load level from tag files you created.

> **NOTE**
> For the general info on **halo3_tag_test.exe**, see the Halo 3 Modding - General Concept doc.

To load your level in **halo3_tag_test.exe**, do the following:

1. In the root folder of Halo 3 Editing Kit (where **halo3_tag_test.exe** is located), create the **init.txt** text file.
2. Open this file with a Notepad, and set your level as the starting level for **halo3_tag_test.exe**, using the game_start <path_to_level_from_the_"tags"_folder> command written in this file.

 For example, in our case, we will need to add the following text to this file:

   ```
   game_start levels\mod_levels\my_level_1\my_level_1
   ```

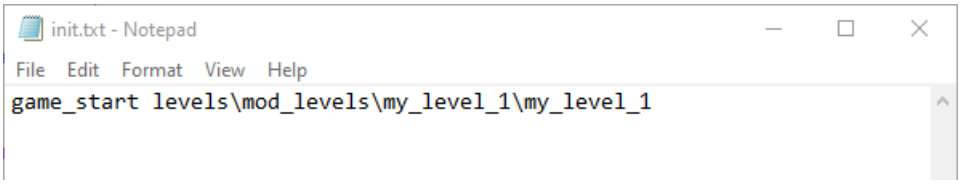


Fig 1. Init.txt with the game_start line added.

3. Save the init.txt file.
4. Now, to load your level, you can simply launch halo3_tag_test.exe in the root folder of Halo 3 Editing Kit. However, in this case, the game will start with the default screen resolution, which can be too large. If you want to lower it, see the next step.
5. The resolution of the game can be changed by -width and -height parameters that can be specified in the command prompt.

 For example, to launch the game in the 1280x720 resolution, you should open Command Prompt, proceed to the root folder of Halo 3 Editing Kit there (see Step #3 for details), and launch the following command in it:

   ```
   halo3_tag_test.exe -width 1280 -height 720
   ```

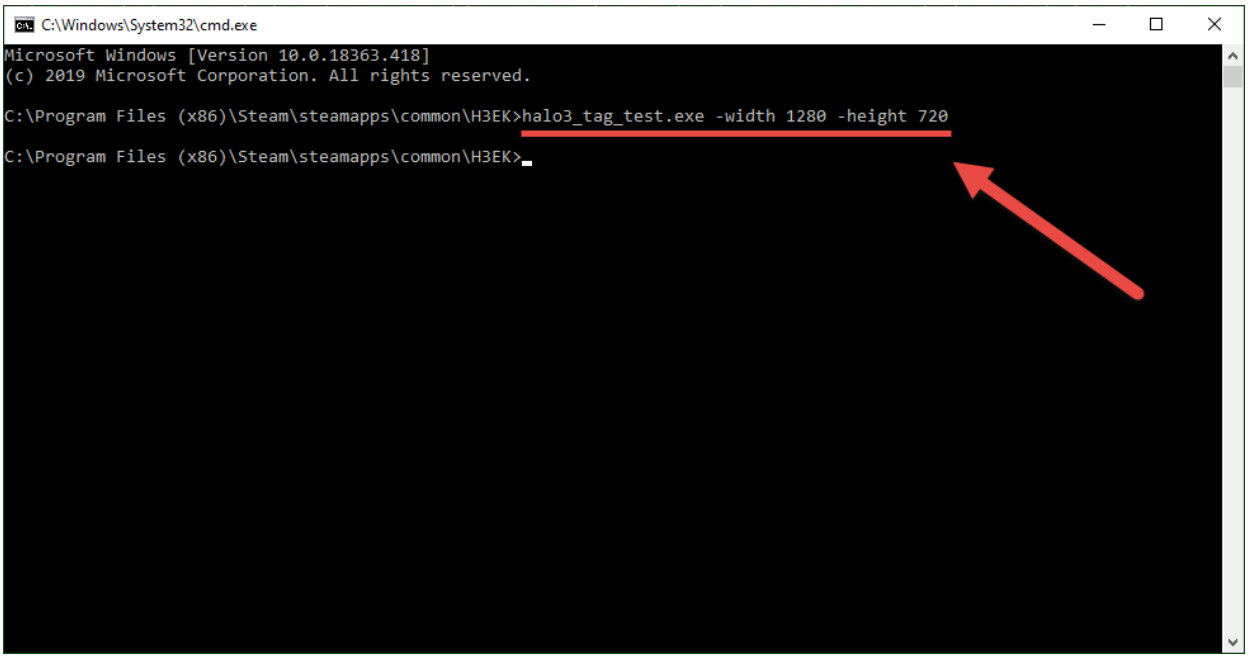


Fig 2. Launch the Game.

> **NOTE**
> You can omit one of these parameters (the default 16x9 ratio will be used in this case).

6. After that, the game will be launched in the specified resolution.

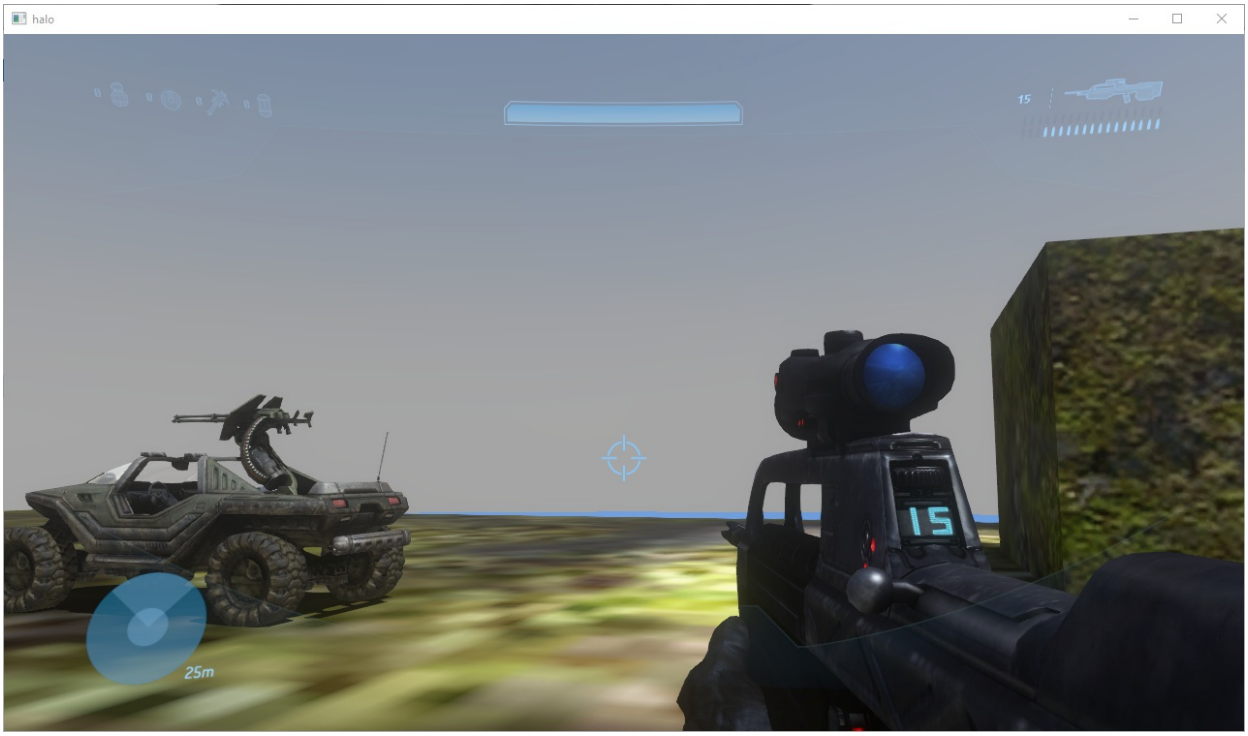


Fig 3. Your playable level!

7. Using standard game controls, you can test your level, ride the Hog, open fire from it, and so on – as if you were in the game.

After that, you can proceed to the creation of the cache file of your level – see the Creation of the MAP file step below.

Quick Start Process Step 9 - Creation of the MAP file

11/23/2022 • 2 minutes to read

Now you have created your level and you can pack it into a MAP (**.map**) file. This file will contain all data of your level in the packaged form: its static geometry, shaders, .scenario file, and all necessary resources and tag files.

MAP files are generated by the **build-cache-file** command of **tool_fast** based on **.scenario** tag files.

To display possible parameters for this command, you can execute the **tool_fast.exe build-cache-file** command in the command prompt

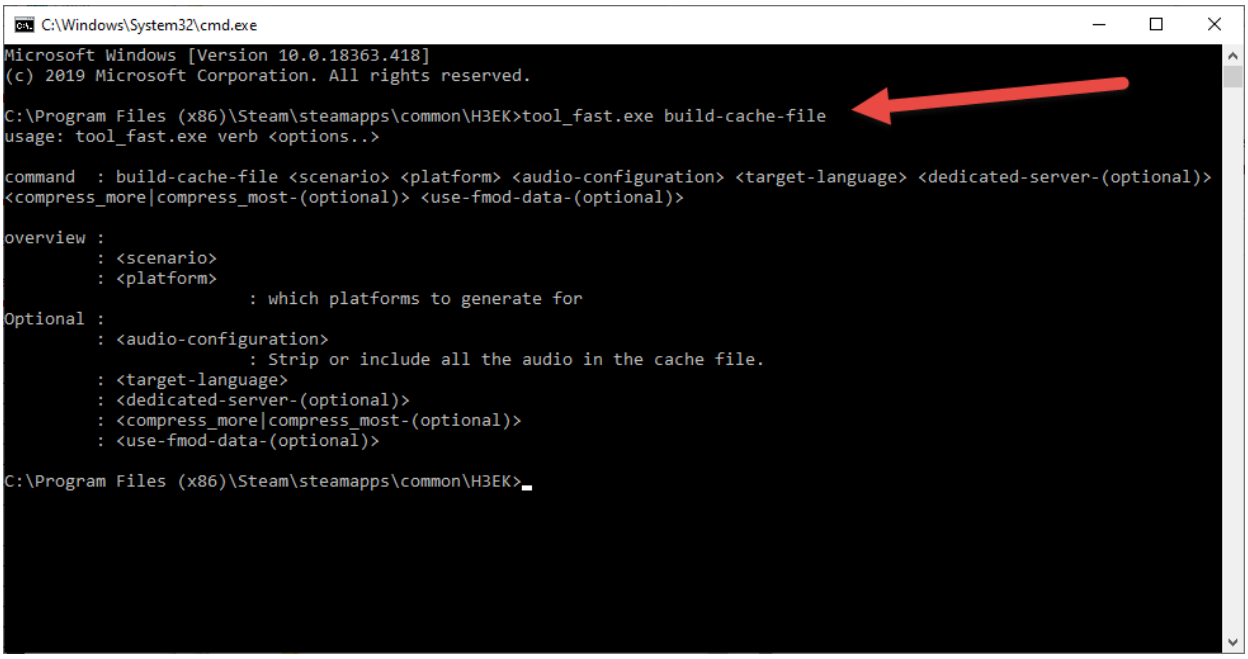


Fig 1. tool_fast.exe build-cache-file without parameters command output.

As you can see, in its simplest form the command requires only the name of the **.scenario** file (without the extension) and the type of the target platform (e.g. **x64** for PC):

```
tool_fast.exe build-cache-file <scenario> <platform>
```

Where, by **<scenario>** here we mean * *<path_to_level_from_the_"tags"_folder>* as before.

> **NOTE**
> As before, you should open Command Prompt, proceed to the root folder of Halo 3 Editing Kit there (see Step #3 for details), and launch this command in it.

For example, in our case:

```
tool_fast.exe build-cache-file levels\mod_levels\my_level_1\my_level_1 x64
```

As before, this will display a lot of debug info (error and a lot of warnings).

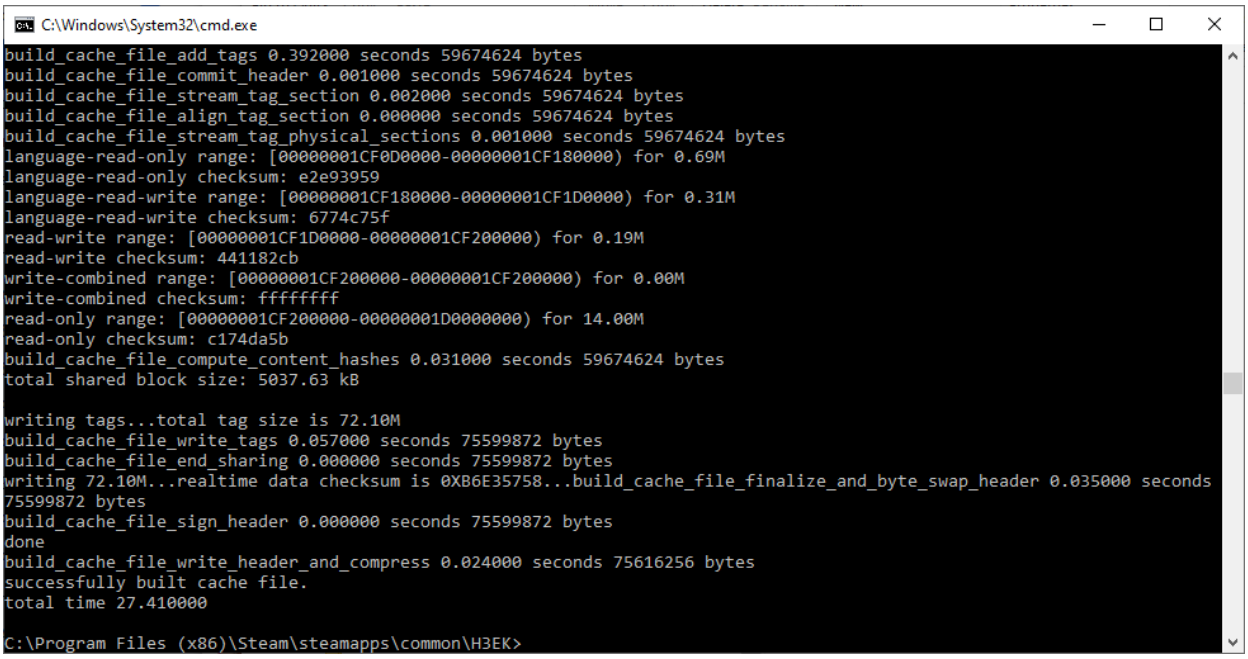


Fig 2. tool_fast.exe build-cache-file with parameters command output.

As a result, in the root folder of Halo 3 Editing Kit, you will find the **maps** folder with the **.map** file of your level.

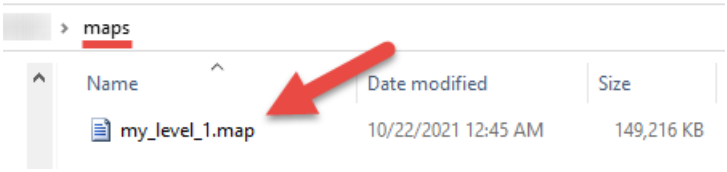


Fig 3. Newly created .map File.

This is the final step of our tutorial. You have passed all the way from the initial geometry in Blender to the final **.map** file and learned the basic modelling pipeline for Halo 3 levels in the process.

Good Job!

After that, you will need this **.map** file to generate a distribution package with the map, add info about the map there (logo, description, etc.), and upload the resulting package to Steam Workshop. After that, other players will be able to download your map and play on it.

However, this process will be performed using a separate tool and it will be described in another tutorial.

Section 1

11/23/2022 • 9 minutes to read

This part of the Designer Boot Camp teaches you how to gather weapons, characters, and objectives to create a playable scenario.

Step 1: Add weapons

Lets add some weapons to the palette.

1. Use Sapien to open scenario your scenario tag
2. Click [edit types] in the Hierarchy View panel.

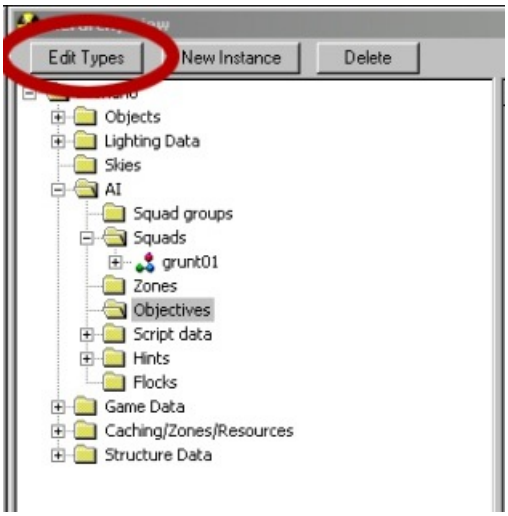


Figure 1 - Edit Types and Hierarchy View.

3. Change the Object Class (in the pulldown menu) to weapons.
4. Click the [add...] button.
5. Double-click pistol.
6. Double-click plasma_pistol.
7. Double-click plasma_pistol.weapon and notice that it added the pistol to the menu below, under Tag.
8. Navigate up two levels to weapons.
9. Double-click rifle.
10. Add the assault_rifle and spike_rifle (follow similar steps to adding the plasma pistol, above).
11. Navigate up to weapons again.
12. Add the brute_shot in the support_low section.
13. Add the gravity_hammer in the melee section.
14. Click [Done].

Step 2: Add characters

Next, add some characters to the palette. The steps below continue the steps to add weapons, above.

1. Change the Object Class (in the pulldown menu) to character.
2. Click the [add...] button.
3. Double-click grunt.
4. Double-click ai.
5. Double-click grunt.character.
6. Navigate up two levels to character.
7. Add a brute.character.
8. Add a brute_chieftain_armor.character.
9. Click [Done].
10. Click [OK].

We added those weapons and characters to the palette so that you can now place them in your scenario. Any weapons, objects, vehicles, characters, equipment and device machines (animated objects) that you want to place in the scenario must be in the palette first.

> **NOTE**
> The larger your palette the longer your level will take to load, so deleting unused items is a good thing.

Step 3: Place grunts

Now it's time to place some grunts.

1. Select Squads under AI in the Heirarchy View.

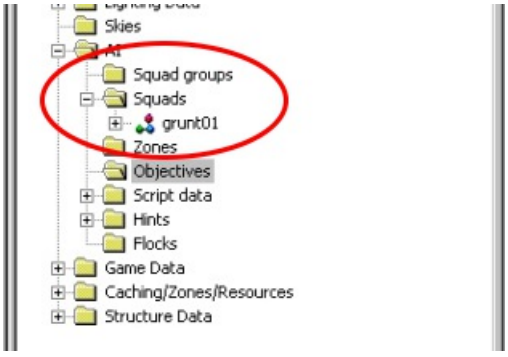


Figure 2 - Squads in Hierarchy View.

2. Click [new instance] and note that a squad named squads_0 is placed under Squads.
3. Select squads_0 and look at your Properties Palette.
4. Rename it to grunts01.
5. Check the flag initially placed in the squads_0 Properties Palette.

> **NOTE**
> Something that is initially placed will spawn when the level is loaded and does not need to be placed via scripting.

6. Change the team to covenant and skip the rest of the pull down options for now.
7. Expand the options for grunts01 in the Heirarchy View.
8. Select the Fire Teams folder under grunts01.
9. Click [new instance].
10. A fire team named none has appeared under Fire teams. Select none.
11. Check your Properties Palette and change the character type pulldown menu to grunt. The name of your fireteam has changed from none to grunt— it will change to the type of AI you place as the default team in the fire team.
12. Change the initial weapon pulldown menu to plasma_pistol.
13. Leave the rest unchanged.
14. Expand the options for your new fireteam (named grunt in the Heirarchy View).
15. Select Starting points.
16. Right-click on the world mesh in the Game window view in Sapien. Notice that a square is placed on the ground that has 3D coordinates, rotation arrows, and text associated with it. The text defines what is spawned in that location. In this case it should be a Grunt with a Plasma Pistol.

> **NOTE**
> The white line on the starting location is the direction the Grunt will face when spawned.

17. You can change this direction with the yellow arrow. Click the yellow arrow and rotate it around.
18. Place eight starting locations.
19. If you need to reposition a grunt, click the white box at the center of its origin and move it around.
20. Select fire team grunt.
21. Change normal diff count from 0 to 6. This defines the number of guys in that fireteam to spawn when the Squad is placed.

> **NOTE**
> You placed eight spawn locations but only have a count of six guys that should spawn. This means that each time the squad is called it will randomly pick six of the eight locations to place a grunt. This is a good way to vary your encounters by having more spawn locations than spawn count causing a different makeup of guys and locations each time.

22. Press alt+r to reset your map. Six grunts should spawn into the level and they should also have green arrows over their heads. Green arrows mean they do not have any firing points to go to. So let's give them some firing points in the next section.

Step 4: Zones and firing points

Before we add firing points, let's get rid of the grunts (for now).

1. Right-click grunts01 under Squads.
2. Select erase squads.

> **NOTE**
> You can select place squads to place the squad as well.

3. Select the Zones folder in the Heirarchy View.
4. Click [new instance] — a zone, named zones_0, is placed under Zones.
5. Expand the options for zones_0 in the Heirarchy View.
6. Select areas under zones_0.
7. To assign the zone to a firing point, select the firing point in the hierarchy and change the area dropdown in the Properties Palette.

> **NOTE**
> Additional firing points created in the Game Window will take on the area of the previously selected firing point.

Step 5: Objectives

This step assigns an objective to the area.

1. Select the Objectives folder in the Heirarchy View.
2. Click [new instance] — new objective called ai_objectives_0 is created under the Objectives folder.
3. Select ai_objectives_0. An entirely new window opens called AI Objectives.

> **NOTE**
> The AI Objectives window is likely quite small at this point. Scale it up so you can see all of the text.

4. Click the [NONE] button next to Zone on the AI Objectives window. A new window pops up called Assign Zone.
5. Assign zones_0 and click [OK].
6. Click [Add] in the AI Objectives window. A spread sheet cell appears in the AI Objectives window. This is a task and it is where the magic happens. More specifically, it's where we assign AI to areas and define the locations the AI should populate during an encounter.
7. Select tasks_0.
8. Rename it task_front.
9. Click on the firing points you created earlier in the Game window..

> **NOTE**
> The firing points are all empty now and become filled when you click them.

10. Select grunts01 under the Squad folder in the Heirarchy View.
11. In the Properties palette change initial zone from none to zones_0.
12. Change initial objective from none to ai_objectives_0.

Step 6: Spawn the grunts

Time to spawn the grunts!

1. Right-click grunts01 under the Squads folder in the Heirarchy View and select Place Squad.
2. Select ai_objectives_0 under Objectives in the Heirarchy View. Notice that the Grunts walked over to your area and are milling around. They mill around because they are in the walk movement speed and are in a passive mode. If they are alerted by an enemy they will enter an alert stance and take up more static positions.
3. Select task_front in ai_objectives_0.
4. Change the movement speed from default to run in the Properties palette. This sets them to have a run movement speed, which will default them into an alerted stance. The grunts will take up positions and look in different directions around the area. If they are not given a direction they will attempt to spread out their facing to cover multiple directions.
5. Select task_front in ai_objectives_0.
6. Hold down the alt key and left- then right-click on the terrain in the Game window. A red line with a red ball is created where you clicked. The red ball defines the forward direction of the line (it's an arrow). The grunts in your task are now facing in the direction the arrow is facing.
7. Hold down the alt key and click with either the left or right mouse buttons and notice the arrow changes with your clicks. The grunts will now face in the new directions.

> **NOTE**
>
> You can delete the direction facing for a task by scrolling down the Properties palette until you find the section called Direction. You will see something like 0. task_direction_block and a [delete] button to the right of it. Click the [delete] button and the red arrow will go away and the Grunts will begin looking all around again (it may take a while for them to reorient themselves).

Step 7: Create more areas

Make two new areas that are a part of zones_0.

1. Select zones_0.
2. You should now have three areas, named: areas_0, areas_1, and areas_2.

> [NOTE] Make sure you have generated the firing points for the areas!

The next steps update your objective with the new areas. Lets use those two new areas to show how to make the grunts fall back from areas_0 to areas_1 and then all the way back to areas_2.

3. Select ai_objectives_0.
4. Click [add] two times to make two new tasks.
5. Rename tasks_1 to task_mid.
6. Rename tasks_2 to task_back.
7. Set task_mid's and task_back's movement speed to run.
8. Select task_front.

9. Hold the control key and press the down arrow — keep holding the control key through the next step. The pink bar to the left of task_front joined task_mid and created a longer bar. don't worry about this yet, we will go into more detail how the different organizations of tasks is accomplished.
10. Keep holding the control key and press the down arrow until task_front is at the bottom of your new tasks.
11. Select task_mid and move it to the middle.
12. Make sure all three tasks have their own separate colored bar to the left of the task.
13. Select task_front.
14. Hold the control key down and press the right arrow key. The task will move to the right and become a child of task_mid.
15. Select task_mid.
16. Hold the control key down and press the right arrow key. task_mid will become a child of task_back. The grunts placed into ai_objectives_0 will enter task_back and then move their way down the task that is closest to the top and then farthest to the right. In this case it should be task_front.
17. If your grunts are not spawned in the map, right-click on their squad and place them.
18. They should run to correct area.
19. Set the Bodies field (found on the task in the AI Objectives window)for task_mid to 4.
20. Set the Bodies field for task_front to 2.

Bodies is a field that causes that task to close down when that number of enemies are killed when in that task. In this case task_front will close down (and not allow new enemies to enter the task when two of the Grunts are killed). Task_mid will then close when two more Grunts are killed. Anyone killed in children tasks are added to the parent tasks. So four grunts need to die in total (two in task_front and two in task_mid) to equal the four bodies to close task_mid.

Step 8: Place a player starting location

Test your magic.

1. Expand Game Data in the Heirarchy View.
2. Select the Starting profiles folder.
3. Click [New Instance].
4. Click the [...] next to primary weapon in the Properties Palette.
5. Navigate to \objects\weapons\rifles\assault_rifle\assault_rifle.weapon and double-click it.
6. Set primaryrounds loaded to 32.
7. Set primaryrounds total to 96.
8. Select Player starting points in the Heirarchy View.
9. Right-click the Game window to place the players starting location on your map.
10. Move the white line to point towards the direction you want them to spawn.

Step 9: Play!

Time to play your scenario!

1. Press alt+r to reset the map.
2. Press tab to change from navigation mode to first person mode.
3. Jump down into the area with the grunts and kill a couple of them They will fall back to the second position.
4. Kill a couple more. They will fall back to the third position.

> **NOTE**
> Select ai_objectives_0 when you are playing the game and watch the tasks update while you kill the grunts and move through the map.

On to Part 2

Congratulations! You have completed Designer Boot Camp Part 1. Now move on to Designer Boot Camp Part 2).

Section 2

11/23/2022 • 12 minutes to read

This part of the Designer Boot Camp teaches you how to set up objectives and scripts to create a playable scenario.

To begin this part of the boot camp, open your scenario.

Step 1: Check your palette

Your palette should have the following in it (steps for adding things to a palette are described in Designer Boot Camp Part 1):

Weapons

- rifle/assault_rifle
- rifle/spike_rifle
- pistol/plasma_pistol
- pistol/excavator
- melee/gravity_hammer
- support_low/brute_shot

Equipment

> **NOTE**
> You will need to navigate from the equipment section back to weapons. Grenades are added in the equipment section of the palette, but are found in the weapons section of the file structure.

- weapons/grenade/plasma_grenade
- weapons/grenade/claymore_grenade
- weapons/grenade/frag_grenade

Characters

- grunt
- brute
- brute_chieftain_armor
- brute_captain

Now you're ready to add more zones, areas, and firing points.

> **NOTE**
> Make sure to add the firing points for each of the areas.

Step 2: Place some dudes

Now it's time to make four squads.

1. Place one squad of four grunts wherever you would like. Arm them with plasma_pistols and name them squads_0.
2. Place one squad of one brute captain and two brutes in a separate area. Give the brutes spike_rifles and the captain a brute_shot. This will be squads_1.

There are two ways to place two different types of characters in the same squad:

- Make two separate fire teams in the same squad
 - One fire team has two brutes and one has the captain. This is the more desirable option of the two, as it lets you create multiple starting locations for both fire teams and still spawn a predictable number of each team — for example one captain and two brutes.
- Make one fire team of brutes
 - Create three starting locations for the brute fire team. Select one of the starting locations and in the Properties palette change the character type from NONE to brute_captain. In this situation if that starting location gets used it will spawn the brute_captain. If you ended up placing six starting locations and one of them was the brute_captain, and you set the spawn count to three, you would only get the brute_captain 50% of the time. This is why the first option is preferable, but this is a good place to set different character variants (like what weapons they are carrying).

3. Place one squad of two brutes. Give them one an excavator and one a spike_rifle.
 - Do this by creating two starting locations for the brutes and set one to have spike_rifle and the other to have an excavator. This is done in the same manner as #2 above, but instead of changing the character type change the initial weapon. This will be squads_2.

> **NOTE**
> If the options in the starting locations are set to none, that AI will use the defaults set in at the fire team level.

4. Place one squad of one brute chieftain armor in different area. Give him a gravity_hammer! This will be squads_3.
5. Place all your squads in the level and see if it works! (If you don't have an objective set up, they will have green arrows over their head. No worries, we will fix that!)
6. Set all of your squads to the following settings in their individual Properties palette:
 - Team set to Covenant
 - Initial zone set to zones_0
 - Initial objective set to ai_objectives_0
 - Leave parent and initial task set to NONE.

Step 3: Set up objectives

Before you begin following the steps below, create an objective (or delete the tasks in the objective if you are using your objectives from Designer Boot Camp Part 1) and set zones_0 to your objective.

We are going to do some more complicated task setups to accomplish the following things:

- Player engages the grunts in the initial area.
- The grunts fall back when three of them are killed.
- The two brutes and captain stay in the pillars area until the three grunts are killed, then they move forward.
- The two brutes and captain fall back when two of them are killed.
- The chieftain rushes the player when the two brutes and captain fall back.
- The remaining two brutes and whatever is left of the remaining squads fight the player from a designated area.

Before we get into that, we need to set up the scripts for your level. When you do some of the magic with the objective system it will export a scripts_fragments file. The fragments file will be created automatically whenever you export scripts, but you will need to link your scenario to it first.

To link your scenario:

1. In Sapien, go to Scenarios at the top tool bar.
2. Select Split Mission scripts.
3. Go back to Scenarios and select Add Mission script.
4. This will bring up a windows navigation tool (that should point to your scripts files in the data directory automatically). If it does, select basic_#yourname#*fragments.hsc and click on open, then select basic*#yourname#*mission.hsc file and click on open. The basic*#yourname#_mission.hsc script is the script you will eventually use to write your own scripts. We will do that later, but we will attach it now to get ahead of the game.

Objectives time!

We want the grunts to do the following:

- Spawn in an initial designated area, move to the area where the player spawns, and fall back to an area designated behind their spawn when three of them are killed.

To do this, follow these steps:

1. Create three new tasks in your ai_objectives_0 (remember to click on the [add] button in the AI Objectives window to create a task).
2. Make one of those tasks a gate task. This is done by checking the gate flag in the properties palette when the task is selected.
3. The task will turn gray. This means that no AI should ever live in this task. It is just used to manage other tasks. This will make sense later, so keep on trucking.
4. Name the gate task grunt_gate.
5. Name the other two tasks gg_grunt_forward and gg_grunt_fallback. You can use whatever syntax you like, but a useful setup is to start the names of your tasks with the initials for any gate tasks they are in. Both of these tasks will be a part of grunt_gate so they start with gg_. Follow this naming convention for now and figure out what works for you later.
6. Make the tasks gg_grunt_forward and gg_grunt_fallback children of grunt_gate. Make sure that gg_grunt_fallback is lower than gg_grunt_forward and is not a sibling (see Figure 1).

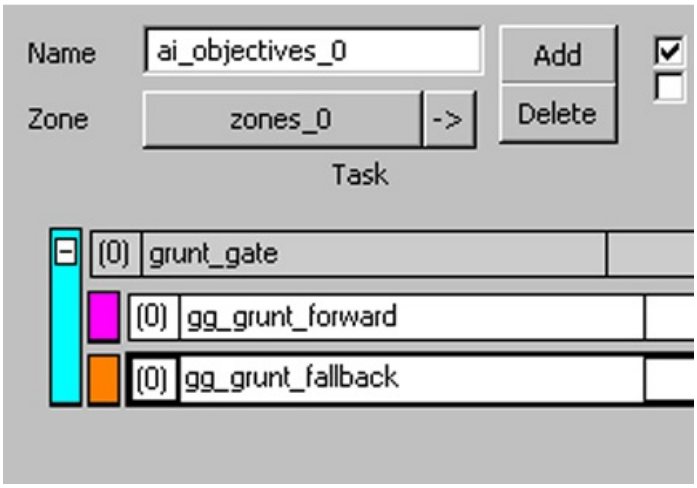


Figure 1 - Proper Hierarchy of gg_grunts.

7. Now set the filter flag for grunt_gate to grunt and check the box to the left of the filter. This makes it so that only grunts can live in this task. No other AI types will ever be allowed in this task. This is a useful way to keep one type of unit separate from the other types.
8. Set the areas for gg_grunt_forward to the areas closer to the player spawn.
9. Set the areas for gg_grunt_fallback to the areas behind the grunt spawn.
10. Set the body count for gg_grunt_forward to: 3. It should look like Figure 2.

> **NOTE**
> Your #fps will likely not add up to the same number of firing points as this example, but it should be in the ballpark.

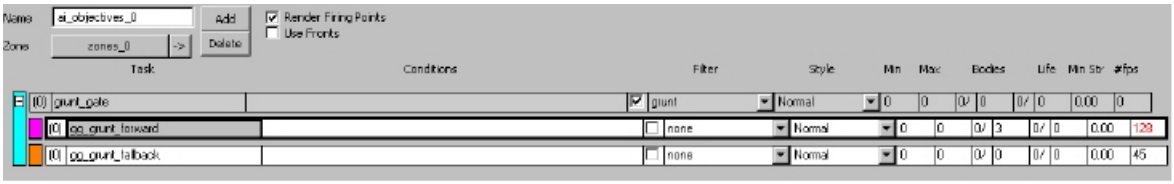


Figure 2 - #FPS Values.

11. Give both tasks a direction, facing towards where the player spawns.
12. Set your tasks to have movement set to run.
13. Reset the level by pressing alt+r, then press <tab> to enter the game in first person mode and watch the grunts. They should spawn and move into the first are gg_grunt_forward and then when you kill three of them they should fall back to gg_grunt_fallback.
14. Now let's set up the first squad of Brutes. This will be the two Brutes with the Brute Captain (squads_1). Squads_1 should follow this layout:

- They spawn in their initial spawn area.
- They should stay in the area nearby.
- They should move into the next designated area when the grunts fall back.
- They should move into the next designated area when their body count is two.

Step 4: Set up squads_1

Squad groups are effective ways of organizing your AI into groups that can be used to filter them into tasks and also are very useful when you get deeper into scripting. For now, we are going to give squads_1 a squad group, which will allow us to give them some tasks that only they can use.

1. Create a squad group by selecting the Squad groups folder in the Hierarchy view.
2. Click the [New Instance] button.
3. A new element appears in the Heirarchy view called squad groups_0. Rename this (in the properties palette) to BruteCaptain.
4. Leave parent set to NONE and change initial objective to ai_objectives_0.
5. Select squads_1 and set parent in the Properties palette to BruteCaptain.
6. Create four new tasks in ai_objectives_0.
7. Rename one of the tasks captain_gate and check its gate flag.
8. Rename the other three tasks the following:
 - cg_start
 - cg_forward
 - cg_fallback
9. Make them children of captain_gate. It should look something like Figure 3:

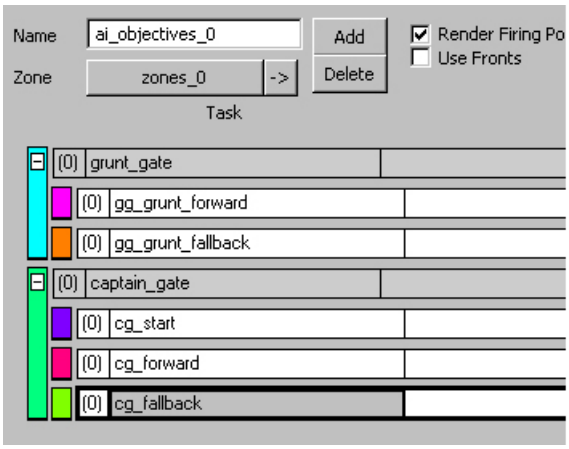


Figure 3 - Captain_gate.

10. Now let's use the squad group we set up earlier. Set captain_gate Squad group filter to BruteCaptain. You will find this in the Properties palette. Now only squads with BruteCaptain as their squad group can enter the gate task (and by definition any children tasks of that task).
11. Set cg_start to match the area you want for enemy spawns.
12. Set cg_forward to match the area you want for them to move to.
13. Set cg_fallback to match the area you want them to fall back to.

Step 5: Your first script

Now for your first foray into scripting. We need the Brutes to stay in cg_start until three or more Grunts die (which forces them out of gg_grunt_forward). We will do that by setting up cg_start to only be active when there are Grunts alive in gg_grunt_forward. Do this with the following script:

```
(> (ai_task_count ai_objectives_0/gg_grunt_forward) 0)
```

Copy that script into the Conditions cell of cg_start. You will need to click on the conditions cell for that task and expand the size of the window.

What that script means:

- All operations (+, -, >, <, =, etc.) are placed at the beginning of a script.
- The two elements after the operation will be evaluated by the script.
- Anything in parenthesis is evaluated as one element.

In this case we are checking that the first element is greater than the second element. Ai_task_count is a script that checks the number of AI living in a task. Ai_task_count is passed the location of an objective/task and evaluates how many AI are in that task. In this case it's the task gg_grunt_forward. So as long as there are greater than 0 AI living in ai_objectives_0/gg_grunt_forward, cg_start will be open. Once there are no longer >0 AI living in gg_grunt_forward (which happens when the 3 body count in that task is reachecg_start will close and the Brutes/Brute Captain will move to the next available task.

> **NOTE**
> Conditions for tasks are initially colored blue. This means that the script has not been compiled for this condition. Once it has been compiled the color of the text will turn black if it compiled properly and it will turn red if it failed compiling properly.

Next, perform these tasks:

1. Set cg_forward bodies to 2.
2. Set the tasks movement to run.
3. Give the tasks direction facings.
4. Compile your scripts by hitting control+shift+c.

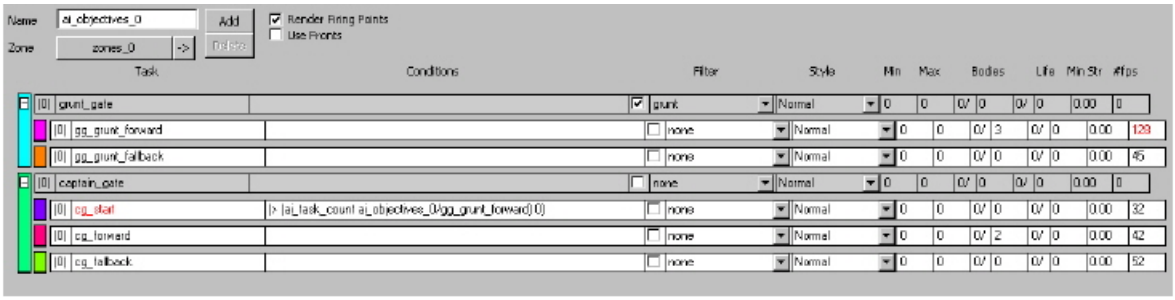


Figure 4 - Compiled Script.

You will notice that cg_start is red. This means that it's closed to entry because the conditions are not met. There are no grunts in your level (assuming you have not set them to spawn automatically), so there is no one in task gg_grunt_forward, which means the script you wrote is not true. Right-click squads_0 and place them. Notice that cg_start turns black, meaning that it's open for AI to enter.

Now lets set up the other two Brutes, the AI from squads_2. These guys are easy— they hang in the back and stay there. We want them to occupy the designated space, and stay there.

To set up the other two brutes:

1. Create two tasks in ai_objectives_0.
2. Make one a gate task and name it brute_gate.
3. Name the other one bg_forward and make it a child of brute_gate.

4. Give bg_forward a run movement speed.

5. Assign the areas you want the squad to move towards to bg_forward.

Thats it! It should look something like Figure 5:

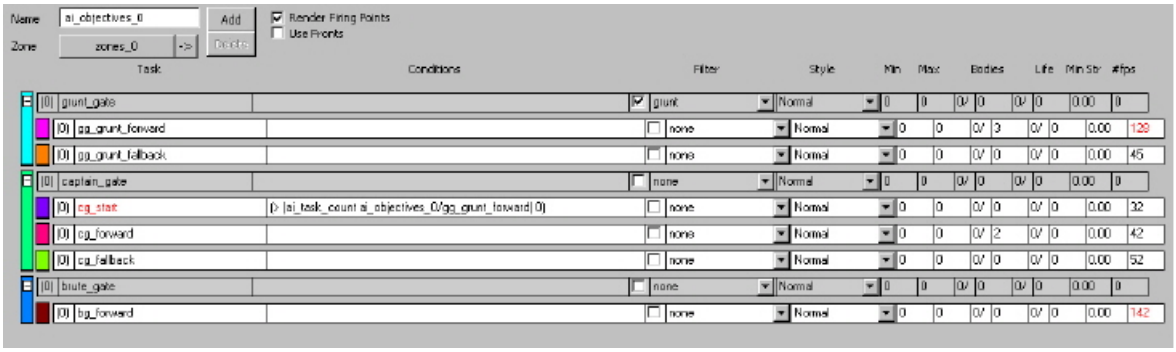


Figure 5 - Compiled Script.

Now for the Chieftain

The Chieftain is pretty easy too— he starts out in the back and waits for the Captain to fall back (who is waiting on the Grunts to fall back, then he charges).

Create three new tasks:

1. Rename one chieftain_gate and make it a gate task.

2. Name one chg_start (we used chg_ in order to not conflict with the cg_ task that refers to captain_gate).

3. Name the last one chg_follow.

4. Make chg_start and chg_follow children of chieftain_gate.

5. Set both tasks to have a run movement setting.

6. Assign the areas to each task.

7. In the Properties palette, set chg_follow to be a follow task, and have it follow the player.

8. Set the follow pull down option to be player. Now anyone in that task will follow the player, and in the case of AI who hates the player, it will follow him and try to kill him.

It should look something like Figure 6:

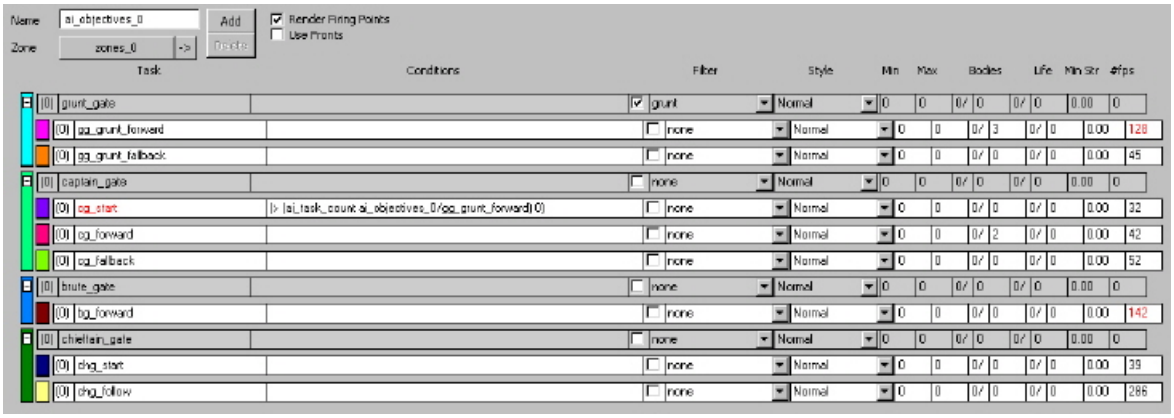


Figure 6 - Compiled Script.

Now try and spawn all four groups, and spawn them in order (squads_0, squads_1, squads_2, squads_3). Notice something bad happens. The chieftain enters bg_forward with the other two Brutes. We don't want that, we want him to enter on his own tasks. We could do this by setting another squad group (like we did for the

captain), but try a new way...

1. Swap brute_gate with chieftain_gate (keeping the children tasks with their respective parents). Do this by selecting chieftain_gate, press the control key, and press the up arrow twice. The two tasks should swap spots.
2. Give chieftain_gate a leader filter and check the flag to the left of the filter.
3. Set Max on chieftain_gate to be 1. Min and Max are ways to control the numbers of AI that can be in a task. By setting chieftain_gate to be 1, we make it so that there and be no more than one AI in chieftain_gate at any time.

It should look like Figure 7:

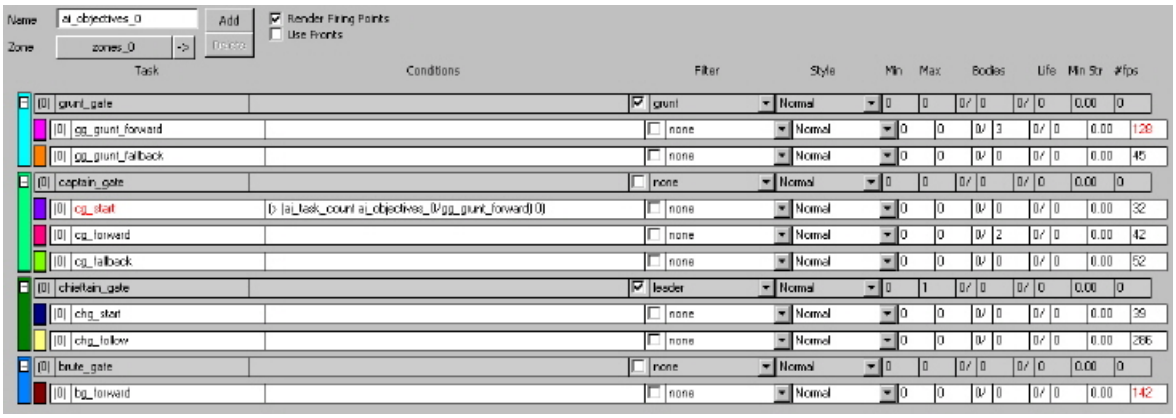


Figure 6 - Compiled Script.

Now try spawning each squad (in order). They should filter into each task properly.

Step 6: Test it out

1. Set all of your squads to be initially_placed.
2. Add a player starting point and starting profiles (both found in Game Data) if needed.
3. Reset your level (press alt+r).
4. Hit tab to enter first person mode.
5. Run around and see what happens. Have the ai_objectives_0 open while you are running around and killing things to watch the tasks updates (NOTE: There is a delay between the AI changing their tasks and the AI Objectives window getting updated. Should be less than a second in most cases.)

Experiment

There are several ways to break this encounter. Play it some and find ways to make the AI behave in ways that you would not expect, or that react in a way that would make them seem smart.

Congratulations! You've completed Designer Boot Camp Part 2!

Halo 3 How-Tos

11/23/2022 • 2 minutes to read

Here is a collection of How-To guides. More information can be found by selecting the topic you're looking for.

Build Cache Files

Guide to building a cache file.

Create Slip Surfaces

A typical Halo environment is rolling terrain with tall bounding walls. Unfortunately the slopes aren't sheer cliffs, nor do they slope upward at an even rate. All too often we discover that a player can exploit a poly or two and run, hop, or otherwise scale these slopes until they are outside of the world. To fix this we have slip surfaces.

Create Soft Ceilings

Soft ceilings are barriers of any type, not just ceilings, that will force the player (but not AIs and vehicles) in the direction of the normal of the triangle if they get past it, soft bouncing them back instead of stopping them cold like a wall.

Executing Multiple Script Commands

Guide on how to execute multiple script commands in a single line.

Find Console Commands

Information on a helpful console command which allows you to find other console commands!

Setup Init.txt

The init.txt file contains console commands or other important information that you want to get read whenever Halo is launched.

Init.txt Scenario Pruning

Information on how to prune assets through the init.txt file.

Multiplayer Map Setup

Overview of how to set up multiplayer maps in Halo 3 using Sapien with some Guerilla tidbits thrown in as well.

Using Portals

This article will show you how to set up and maximize portals as well as more information on what they are.

Water Physics

Information and guide on how to set up and use water physics volumes.

Weapon Overview

This article contains information about the properties of the magazines for each weapon.

Build Cache Files

11/23/2022 • 2 minutes to read

To build cache files, do the following:

1. Open up a command window and navigate to the root folder of the branch you're working in inside the depot (\halox\main, for example).
2. Type Tool **build-cache-file path_to_your_map** (levels\multi\zanzibar\zanzibar, for example). This builds a .map (cache) file and places it in a folder in your root branch directory called Maps (\halox\main\maps in most cases).

If you would like to sync the cache files to your Xbox as you build them, you can use the **build-cache-file-sync** command (the only difference is that it syncs the files to a tags build on your xbox after building them).

Create Slip Surfaces

11/23/2022 • 3 minutes to read

A typical Halo environment is rolling terrain with tall bounding walls (see Figure 1). Unfortunately the slopes aren't sheer cliffs, nor do they slope upward at an even rate. All too often we discover that a player can exploit a poly or two and run, hop, or otherwise scale these slopes until they are outside of the world. To fix this we have slip surfaces. You can make any Bungie Material (BM!) into a slippery one by clicking the Slip Surface check-box in 3ds Max (see Figure 2).

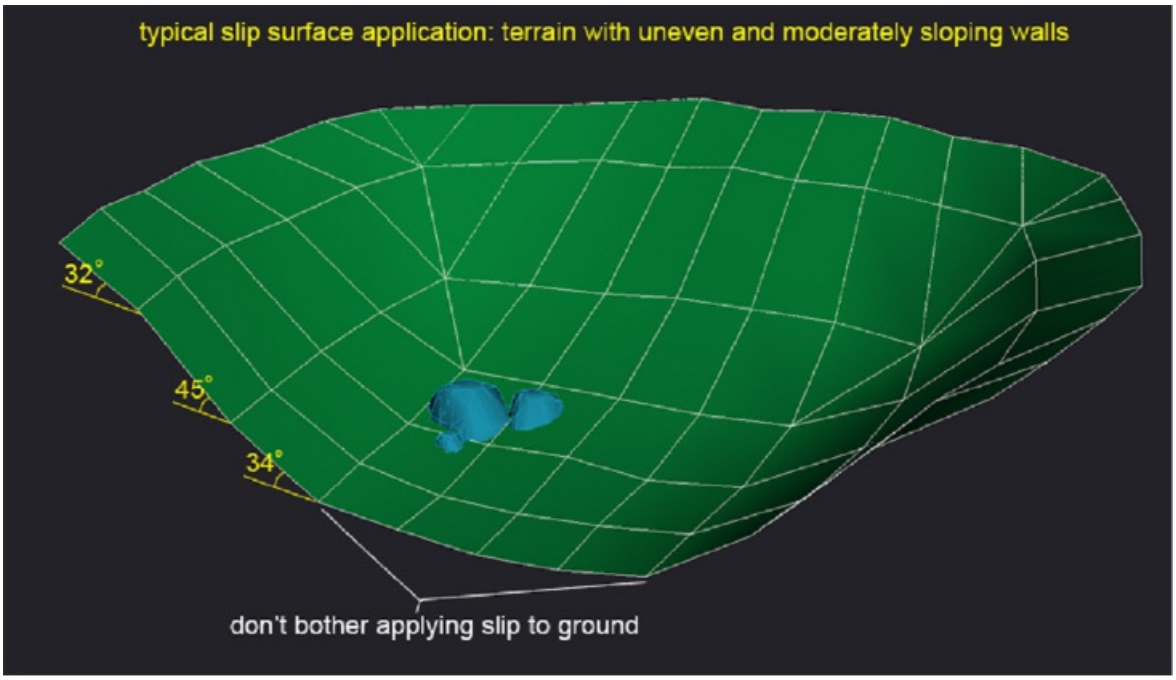


Figure 1 - Uneven and sloping walls

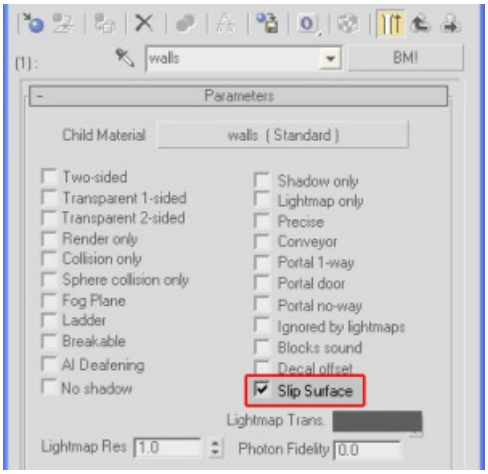


Figure 2 - Slip Surface Checkbox

Although it may be simpler to apply a single shader to the entire terrain, you should avoid applying slip to areas that you intend the player to travel (this eliminates the need to debug staircases, one-foot drop-offs, etc.). In our example, this means creating two material slots for our terrain: one for the ground and another for the walls.

Also, the engine will evaluate every poly you designate as a slip surface and determine if it is actually suitable for slipping. This means you can quickly grab all the sloped surfaces and make them slip without worrying too much about the occasional flat spot. This also means you can apply slip to poops and the system will determine

which faces should be slippery based on each instance's particular orientation.

> ![Note] The current standard is 35 degrees from horizontal. Anything above that will become slippery.

Once you import your level, you can check out your slip surfaces in the game through the debug menu: debug -> environment -> debug_structure_slip_surfaces. Any surface marked with a slip material will have colored edges. Red means the face was too shallow to make slippery, green means it's slippery. Figure 3 is a 3ds Max screenshot approximating what you'll see in the game engine.

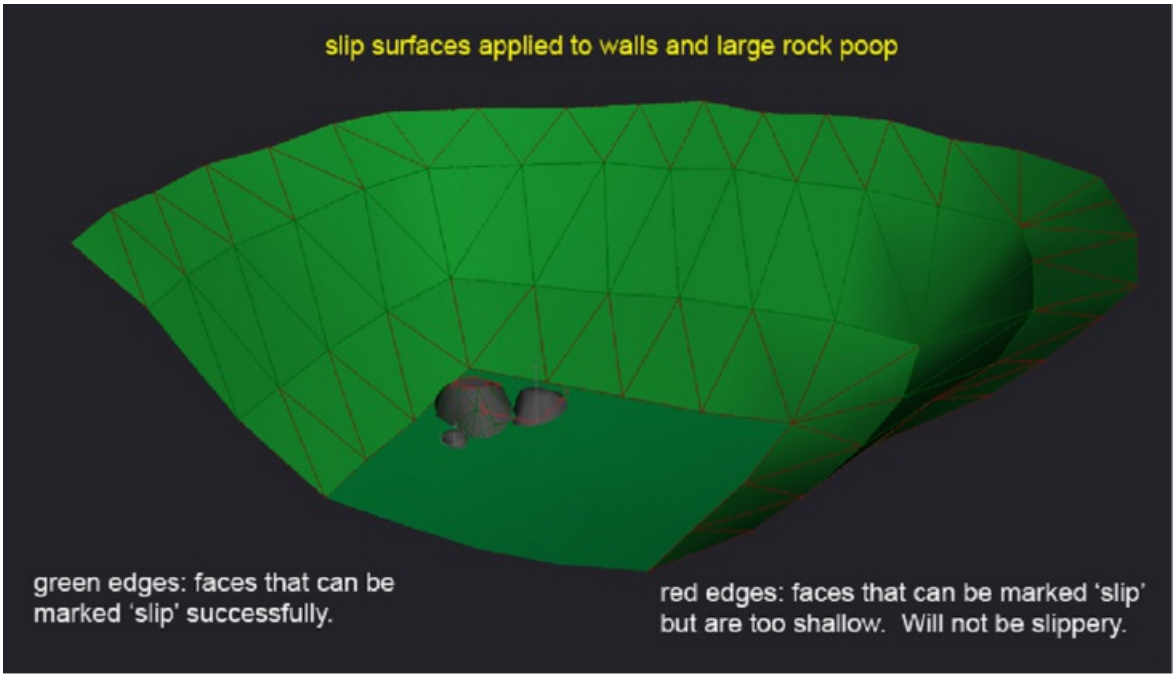


Figure 3 - Edges that can and cannot be made slippery

Players can become trapped or defeat your slip surfaces if there are poops or jagged crannies in your sloped walls. If wedged between slip surfaces the player may become immobile, especially if he's in a vehicle. If non-slip faces (like tree root and rock poops) exist among slippery ones, then players can find ways of traveling along these, thus defeating the purpose of slip surfaces altogether. In these cases you will have to make these offending polygons render-only or slippable (slip surface material and all exposed faces are steeper than 35 degrees). If this won't work for you, then placing a soft ceiling around the offending area will help.

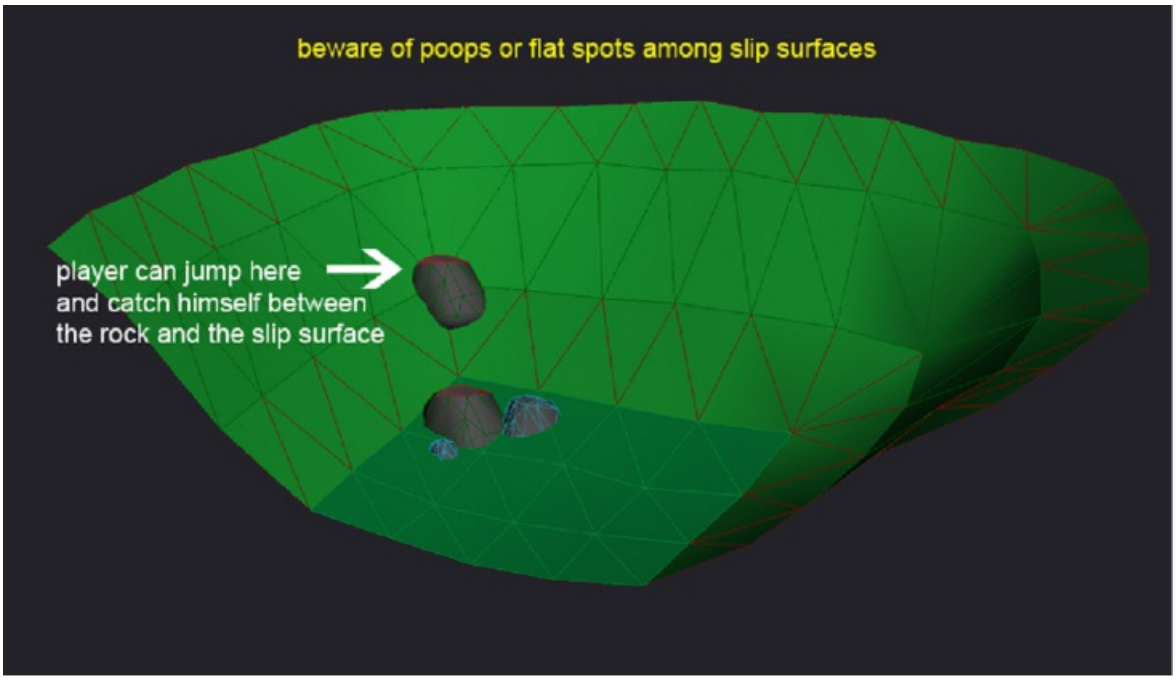


Figure 4 - Flat spot among slip surfaces

NOTE

- Bipeds can not form ground planes on these surfaces and vehicles can not drive on them.
- Currently if you make one of these surfaces more shallow than 35 degrees we flag it as an error.

Create a Slip Surface

For Halo 3 has the ability to update and add Slip Surfaces to the Structure Design file, thus eliminating the need to create lightmaps every time we change a surface to be slippery. The new slip surfaces only work on angles greater than 35°, same as the other way of doing slip surfaces.

To use the new slip surfaces, the level must have a structure design file. This is located in the structure_design folder in the level name folder. You should Xref the scene from the actual file.

To create a new slip surface

1. Create a plane above the surface that the player will intersect with. This plane must have material type +slip_surface:name.
2. After you've created the plane and added the material type, you need to export the structure_design to an ASS file.
3. Import the ASS file as structure design from inside Guerilla.
4. Open the structure file and add any structure_design files to your BSPs.
5. Save and Xsync.

Create Soft Ceilings

11/23/2022 • 4 minutes to read

Soft ceilings are barriers of any type, not just ceilings, that will force the player (but not AIs and vehicles) in the direction of the normal of the triangle if they get past it, soft bouncing them back instead of stopping them cold like a wall. In this way you can foil players' efforts to get into an area of the map you don't want them to enter. Basically it's just like a vehicle ceiling from Halo 2 except it can be any arbitrary geometry and will also work on bipeds.

Defining Soft Ceiling

Name a material +soft_ceiling:name and it will form a soft ceiling in the game

> **NOTE**
>
> - You can turn off a set of these surfaces in the game with the 2 argument hs function (soft_ceiling_enable name boolean).
> - However before you turn off one you must create an entry in the soft_surface tag block of the scenario. This will also enable you to disable it for bipeds and/or vehicles.
> - Also these surfaces need not be manifold or connected, basically you can build them out of any arbitrary triangle soup.

Soft Ceilings in Depth

Where they live

Soft ceilings are now kept in the Structure Design file. The Structure Design file is a 3ds Max file which contains soft ceilings, slip surfaces, and soft kill surfaces. The Structure Design 3ds Max file and ASS file are stored in the /structure_design directory off the level_name directory.

Example

Here is our typical starting point: encounter spaces connected with a transitional zone, in this case, a tunnel. Unfortunately, these walls are low enough that a player can grenade-jump out of bounds (see Figure 1). Fortunately, we can use soft ceilings to keep the player in bounds. Soft ceilings bounce a player back rather than stopping him dead like sphere collision only surfaces. Plus they don't generate a collision sound or effect when hit.

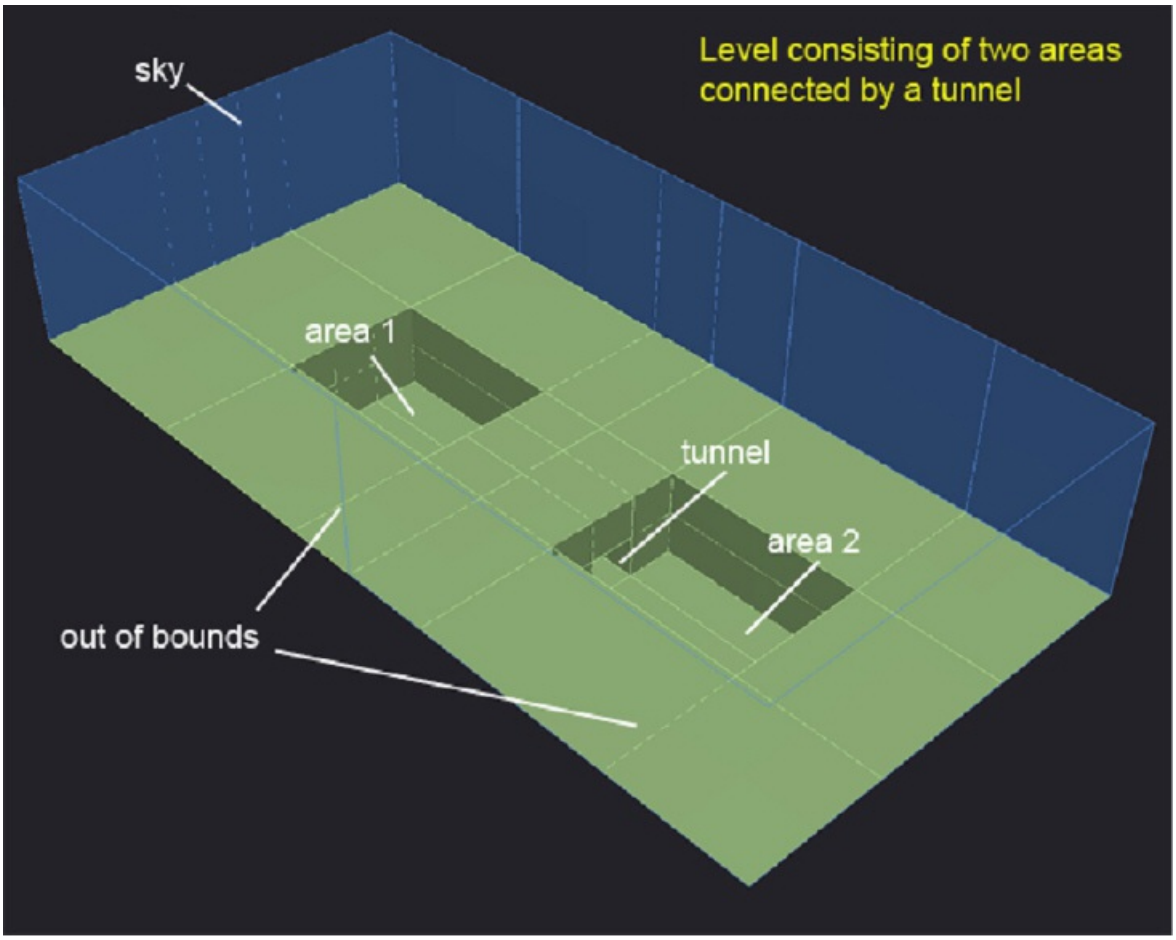


Figure 1 - Connecting Tunnel

Soft ceilings are pretty simple to use, but they must obey special rules when used in multiple bsps, so let's divide this level into two, for the sake of example (see Figure 2).

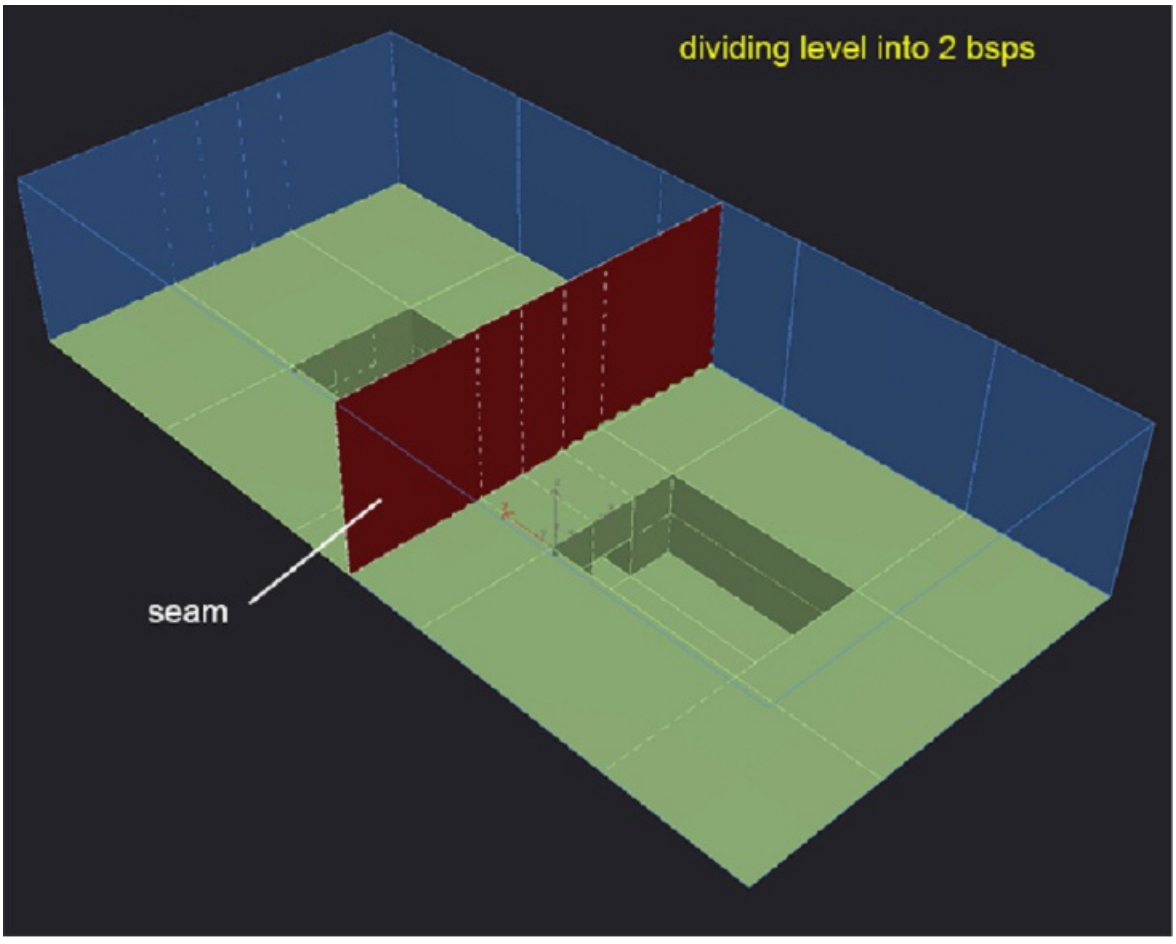


Figure 2 - Divided level

Here we've placed soft ceiling geometry. The face normals point **into** the play area (they're shown double-sided here). Like portals, an edge must connect to another soft ceiling edge, embed into the bsp, or extend through the sky. Also note, although called "ceilings" there is no geometry in this example that acts as a literal ceiling or cap. Instead we have "soft walls." This is because overhead soft ceilings tend to throw the player around when he jumps or explodes into them.

In 3ds Max, create a material sub-object with the following naming convention and apply it to the soft ceiling:

```
+soft_ceiling:<name>
```

The *name* used is the handle the designer can use to turn the soft ceiling on and off. If you need several soft ceilings, simply create more sub-objects with different names. You use this feature in situations where the player needs to be limited during game play, but goes through the soft ceiling during a cut scene.

Since we're going to be exporting 2 bsps, we need to make the soft ceilings in 2 pieces. This is important because, unlike bsp geometry, soft ceilings are not automatically truncated at the bsp seams (see Figure 3).

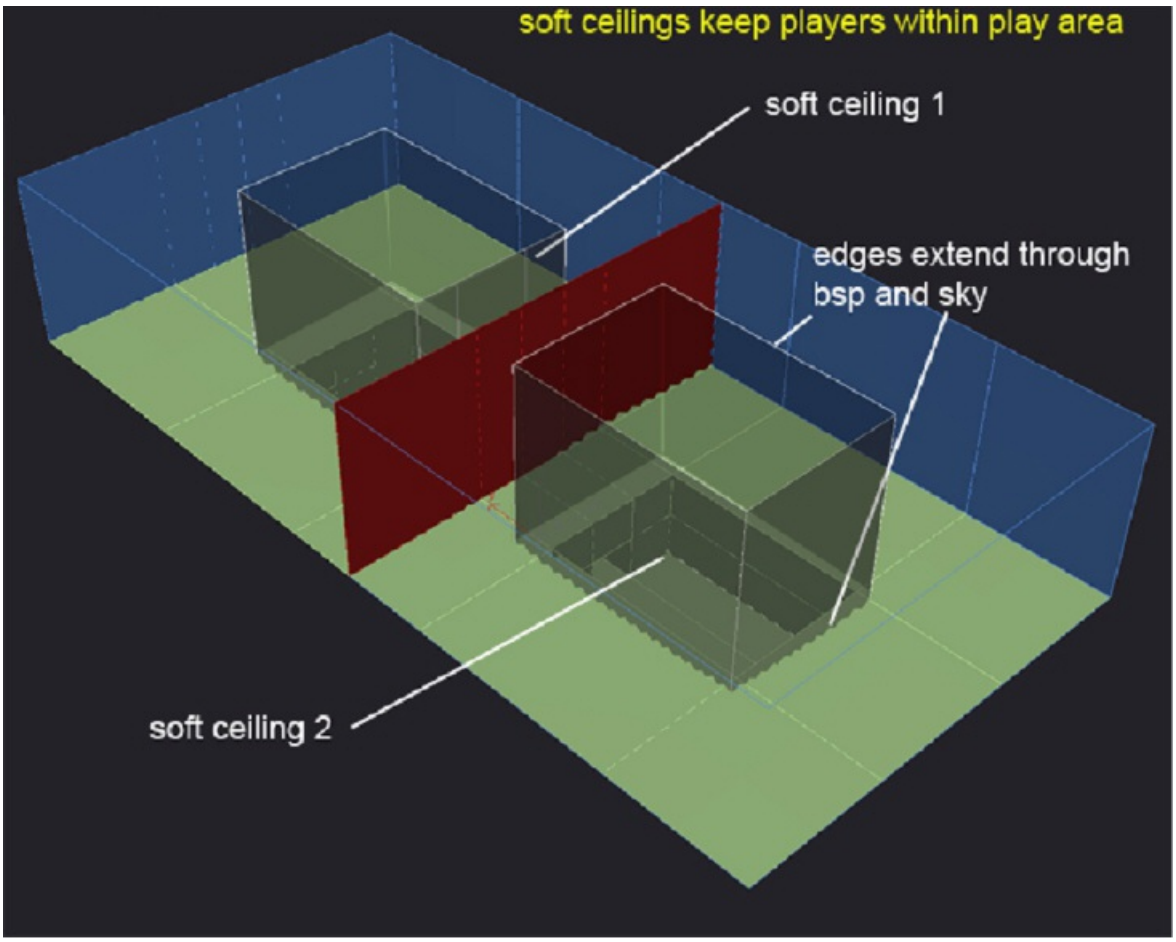


Figure 3 - Soft Ceilings

In Figure 4 you can see what the soft ceiling geometry looks like by itself. Note that we connected the two main volumes of soft ceiling space with a tube of additional soft ceiling material. We have to do this in caves and tunnels because without a continuous connection between the edges near the tunnel, the engine would become confused when you walked "behind" one or both of the embedded soft ceiling edges in that area. None of this tunnel ceiling will be exposed, it will be completely hidden behind the bsp tunnel. Figure 4 shows this more clearly.

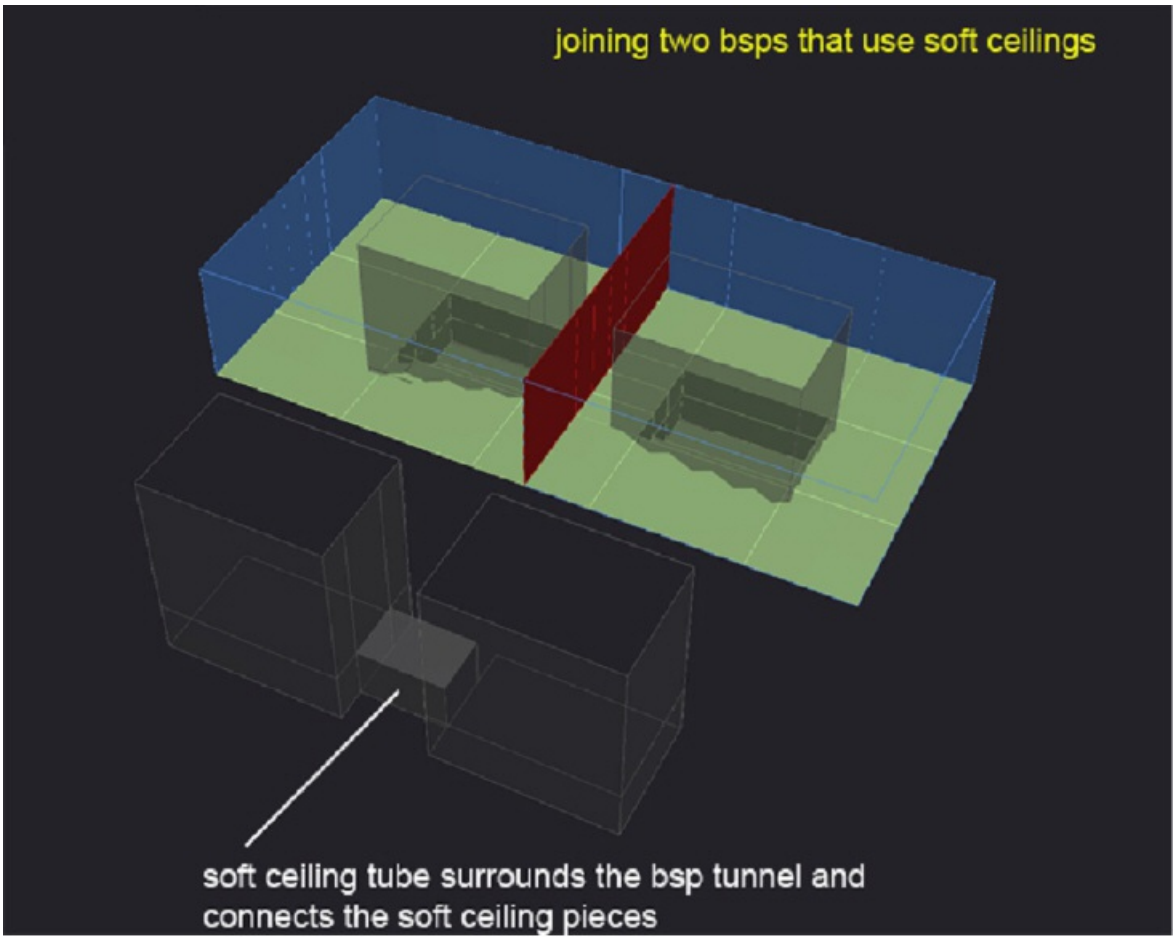


Figure 4 - Two BSPs that use soft ceilings

Figure 5 shows how the connecting soft ceiling surrounds the tunnel geometry. Also note that it's acceptable for soft ceilings to overlap a bit. Once exported, there will be two coincident soft ceilings here, but so long as they're similar and relatively small, you should be fine.

A situation you want to avoid is accidentally exporting all of the soft ceilings into some or all of the bsps. This will create duplicate soft ceilings (which are not reported as errors) that can complicate bug-fixing. You will know you made this mistake if you change a soft ceiling, export/import/sync, and see no changes. This is because there is a duplicate soft ceiling loaded into an adjacent bsp, which you haven't reimported yet. Until you eliminate the duplicate soft ceiling from all the loaded bsps you may not see the correct results.

You can examine your soft ceilings with the command debug_structure_soft_ceilings 1 in Sapien (the recommended way) or use the debug menu on your dev kit: debug -> environment -> debug_structure_soft_ceilings. Blue faces are pointing toward you, red faces are pointing away (see Figure 5).

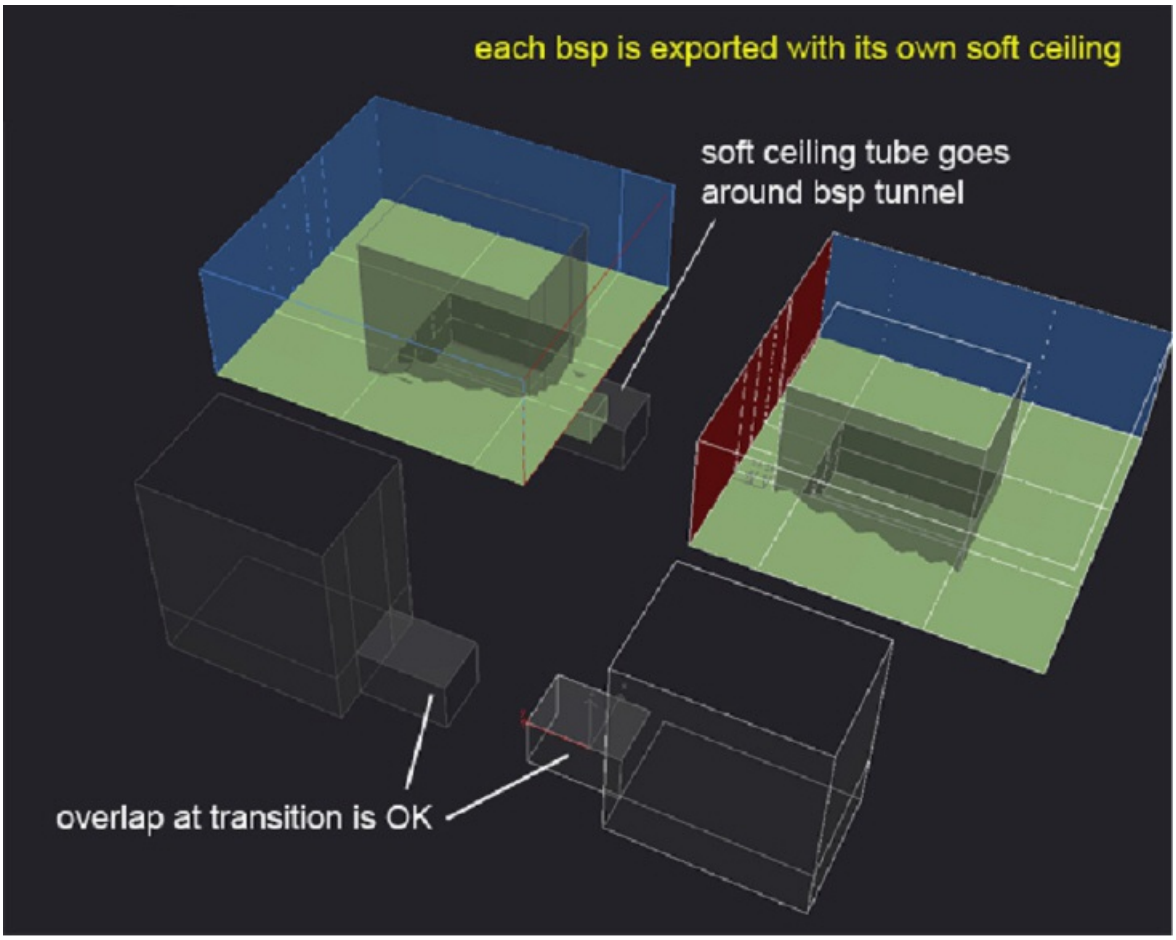


Figure 5 - Each BSP with its own soft ceiling

Examine ceilings near overhangs carefully— once they're embedded into the bsp they shouldn't re-emerge inside the play area (see Figure 6).

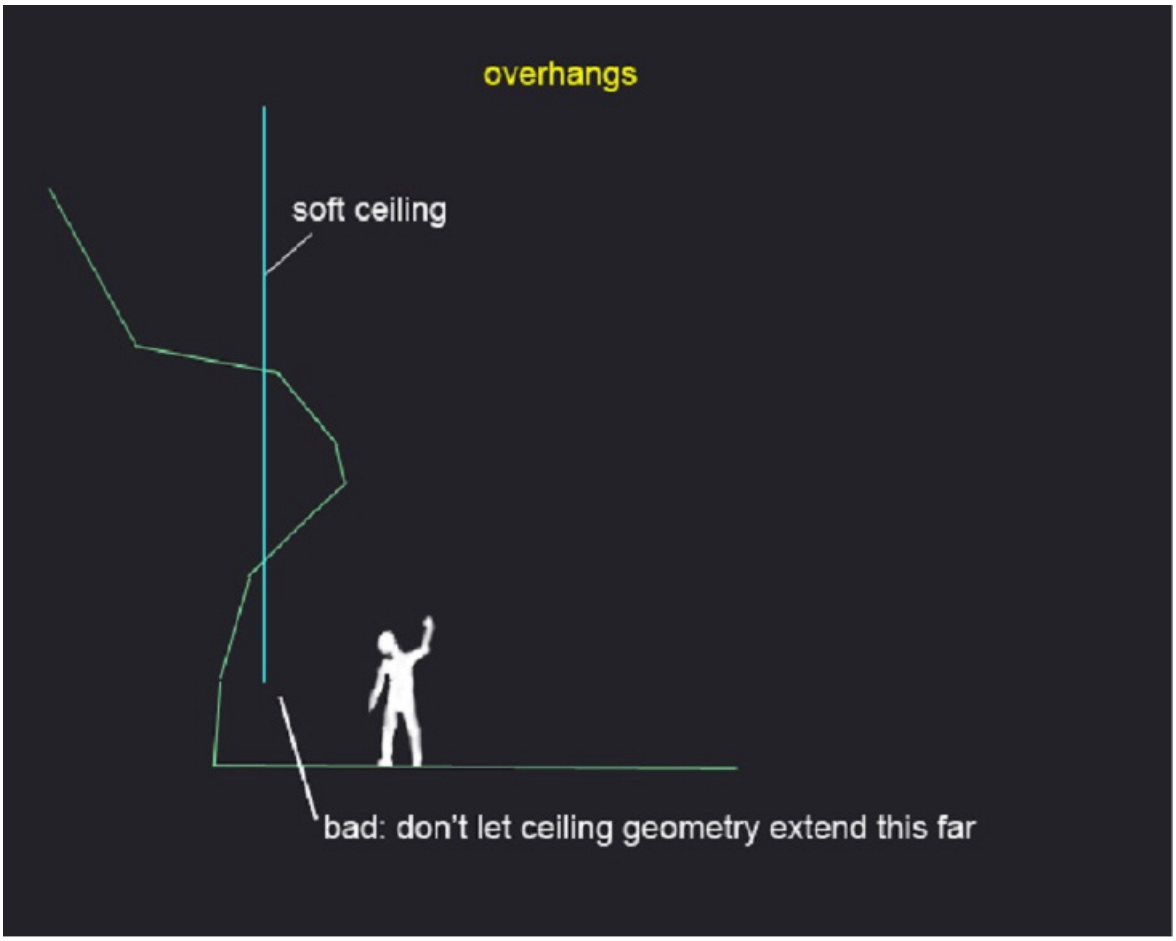


Figure 6 - Overhangs

Instead, soft ceilings should continue well below and away from the intended play area. This prevents players from getting too close or behind a ceiling edge.

Also, soft ceilings should be placed near a steep part of the overhang. If the player ever gets up there, he'd be pushed onto a steep or slippery surface and continue to fall back into the play area (see Figure 7).

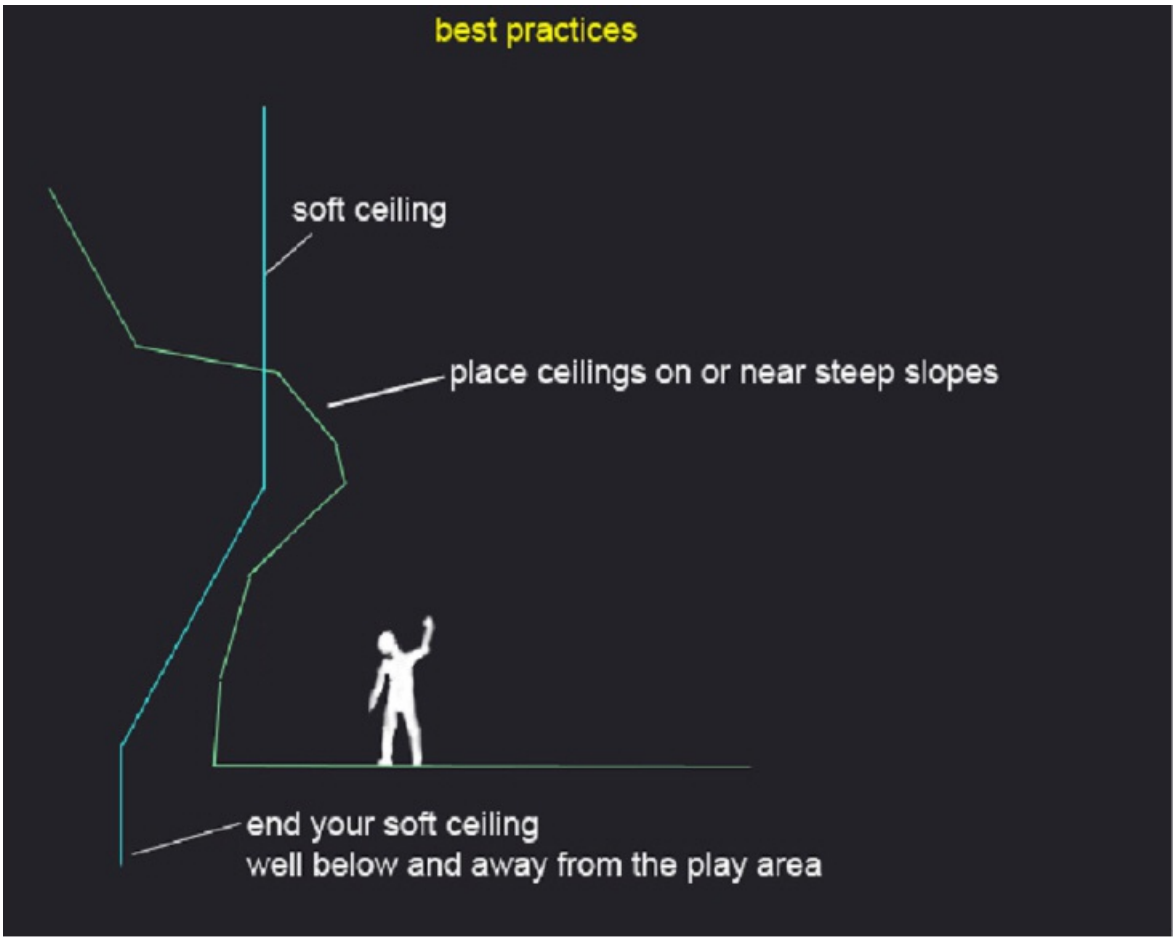


Figure 7 - Best Practices

Executing Multiple Script Commands

11/23/2022 • 2 minutes to read

Executing multiple script commands in a single line

It is possible, and quite handy, to execute multiple script/console commands within a single line. Here's how to do it:

1. Wrap each command in parentheses. For example:
 - (game_speed 0.5)
 - (print "hello")
 - (drop "objects\vehicles\ghost\ghost")
2. Put all the commands in a single line:
 - (game_speed 0.5)(print "hello")(drop "objects\vehicles\ghost\ghost")
3. Finally, wrap the whole line with a (begin <commands>) statement like so:
 - (begin(game_speed 0.5)(print "hello")(drop "objects\vehicles\ghost\ghost"))
 - Don't forget the parentheses at the end!

Find Console Commands

11/23/2022 • 2 minutes to read

There is now a helpful console command which allows you to find other console commands!

- find <string>

For example, typing find physic from the console would show you Fig 1:

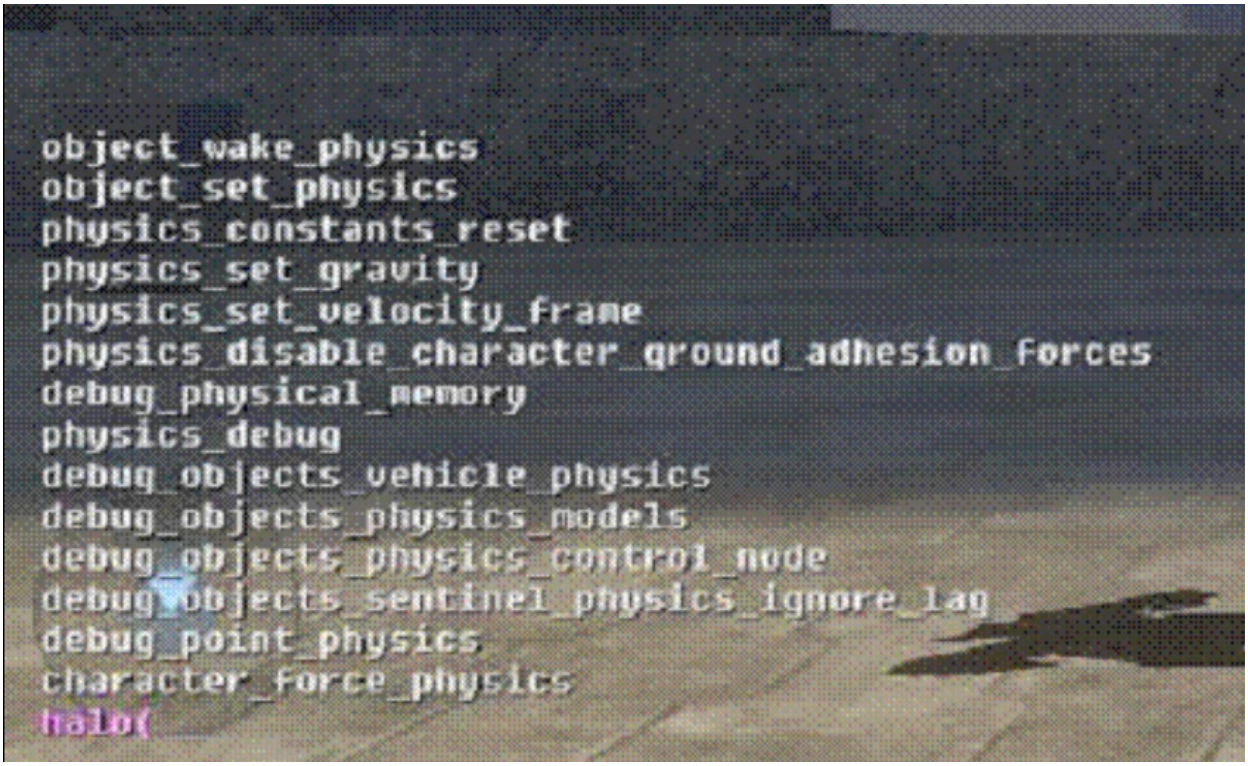


Fig 1 - Find command result

Setup Init.txt

11/23/2022 • 2 minutes to read

The init.txt file contains console commands or other important information that you want to get read whenever Halo is launched. The file itself resides in the root folder of your Halo build on your Xbox and, if you have a tags build, in your \halox\main folder (where it gets synced from whenever you Xsync).

Setup

1. Open Notepad.
2. Fill up the text file with useful or necessary console commands (see below for some examples).
3. From the File Menu, select Save As.
4. In the Save dialog, navigate to \halox\main (or the root folder of Halo if you have your Source Depot client mapped differently).
5. Name the file init.txt and click Save.
6. Use Tool to Xsync, the init.txt file will automatically be copied to your Xbox.

> **NOTE**
> If you aren't using a tags build, you can manually copy your init.txt file to the root folder of your Halo 3 build

Useful Commands

You can place any console command inside the init.txt file and it will be set on launch of the game. If you want to place comments in the init.txt file, use a **; semi-colon** character.

Here are some example commands you might find useful:

- map_nato_map\— Instantly launches into the map specified without making you navigate through the UI
- error_geometry_hidme \path_e_all— Hides all of the error geometry text and numbers for increased performance and easier viewing.

Init.txt Scenario Pruning

11/23/2022 • 2 minutes to read

The game has the capacity to prune assets through init.txt. The functionality essentially converts regular scenarios into empty and empty_cinematic scenarios at runtime.

Commands and Definitions

prune_scenario_force_solo_mode on

- Forces the map to be loaded in solo mode even if it is a multiplayer map. The game will look for a solo map spawn point.

prune_scenario_for_environment_editing_keep_cinematics on

- Same as prune_scenario_for_environment_editing, with the inclusion of cutscene flags and cinematics.

prune_scenario_for_environment_editing

- Removes everything but the environment: Weapons, vehicles, bipeds, equipment, cinematics, AI, etc.

prune_scenario_force_single_bsp_zone_set

- Removes all but the first bsp from the initial zone set. Ensures that the initial zone set will not crash on load due to low memory.

prune_scenario_add_single_bsp_zones:

- Add single bsp zone sets (for each bsp) to the debug menu.

prune_scenario_lightmaps

- Loads the scenario without lightmaps; same as dont_load_lightmaps.

prune_global

- Strips all player information, global materials, grenades and powerups from the globals tag, as well as interface global tags. Same as scenario_load_fast. Don't use this if you need the game to be playable.

prune_global_material_effects

- Loads the scenario without material effects; same as dont_load_material_effects.

prune_global_dialog_sounds

- Removes all dialog sounds referenced on the globals tag. Same as editor_strip_dialogue_sounds.

prune_global_keep_playable

- Strips player information (keeping one element), global materials, grenades and powerups. Keeps the game playable; same as scenario_load_fast_and_playable.

Suggested Configurations

The following table provides suggested configurations for different types of users. You may copy these commands directly into your init.txt file.

DESIGNER	ARTIST	CINEMATICS
prune_global_keep_playable on	prune_scenario_for_environment_editing on	prune_scenario_for_environment_editing_keep_cinematics on
prune_scenario_lightmaps on	prune_scenario_force_single_bsp_zone_set on	prune_scenario_force_single_bsp_zone_set on
	prune_scenario_add_single_bsp_zones on	prune_scenario_add_single_bsp_zones on
	prune_global on	prune_global_keep_playable on

NOTE

Effects should still use the empty_cinematic scenarios because they contain custom scripts.

Multiplayer Map Setup

11/23/2022 • 27 minutes to read

This is an overview of how to set up multiplayer maps in Halo 3 using Sapien with some Guerilla tidbits thrown in as well.

Where to Start

Where to Start First things first, make sure you have your scenario open in Sapien. You also need to have at least the scenario tag checked out, but you'll also want to check out the resources of whatever you are working on. Ensure this directory writable: tags/levels/multi/snowbound.

Don't worry about the changes you make to the scenario on your machine, you can always Get Latest on the entire directory to restore the scenario to the state it was at before you started working on it.

Using Snowbound as our example, make sure you have the scenario tag (levels\multi\snowbound\snowbound.scenario), and the scenery resource (levels\multi\snowbound\resources\snowbound.scenario_scenery_resource).

The scenery resource is what allows you to place all scenery objects and crates (spawn points, respawn zones, flag spawn object, etc.), or basically all the design setup for MP maps apart from weapons, vehicles, grenades, and equipment. You'll be doing most of your work, in terms of sheer map setup, in the scenery resource tag.

If this is a new map, you'll also have to make sure that it will load as an MP map, and the first thing you'll need to do is open the scenario tag in Guerilla. At the very top of the tag, make sure the type is set to multiplayer.

Next, you'll want to scroll down about 50% of the way down the tag until you find a chunk called PLAYER STARTING P.

Add a new starting player then set the following options:

- primary weapon Assault Rifle (objects\weapons\rifle\assault_rifle.weapon)
- primaryrounds loaded = 32
- primaryrounds total = 96
- starting fragmentation grenade = 2.

This will allow you to spawn in the level with an AR and two frags.

Populating the Map

To do this, in your Hierarchy view click Edit Types. A dialog window will pop up, and since we'll want to add spawn points first (you can't spawn into the map without them), we'll start there. This method is how you'll eventually add everything in the map, but we'll start simple.

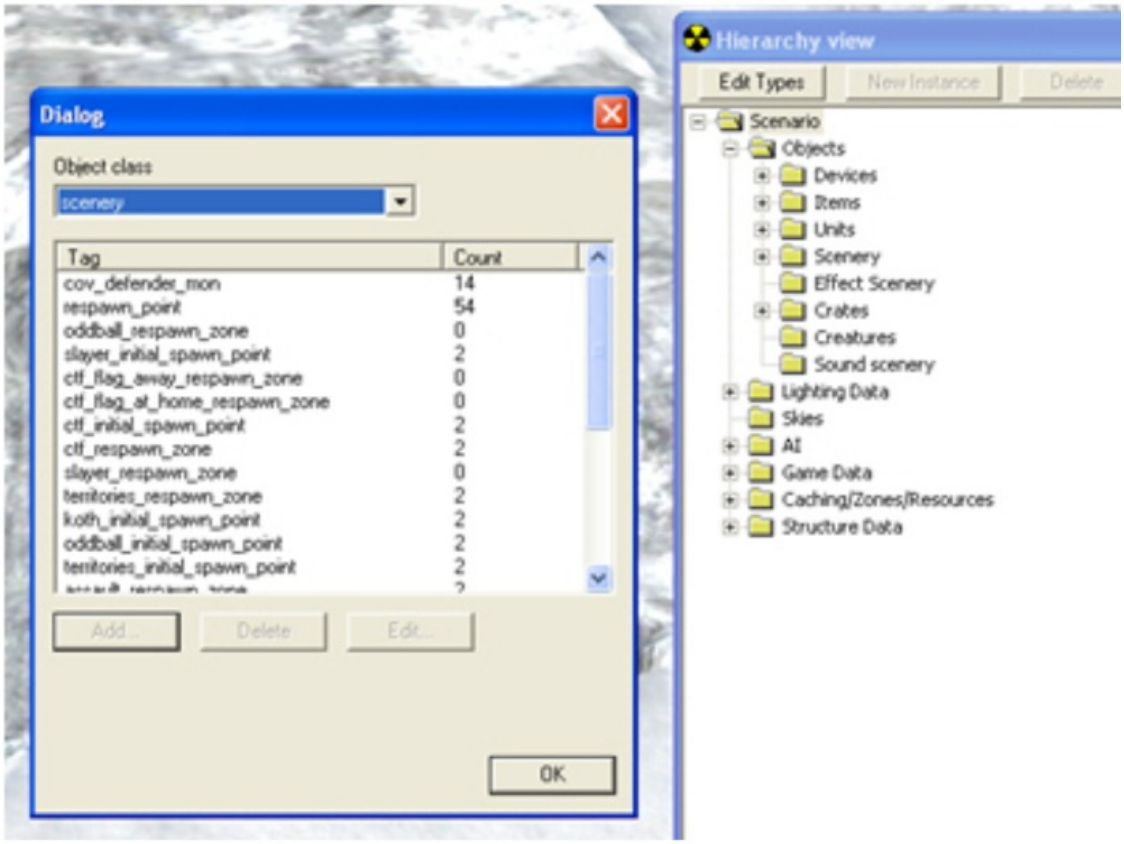


Figure 1 - Edit Types dialog

Under Object class make sure scenery is highlighted then click the Add button.

Navigate to objects\multi\spawning\respawn_point.scenery and add the respawn point. To place one down, follow these steps:

1. In Hierarchy View click the + by the Objects folder to show its subfolders.
2. Highlight Scenery.
3. Go to the Game window and right click— this will place down a blank node.
4. Go to the Properties palette type and choose respawn_point.

You'll see that empty node become a respawn point.

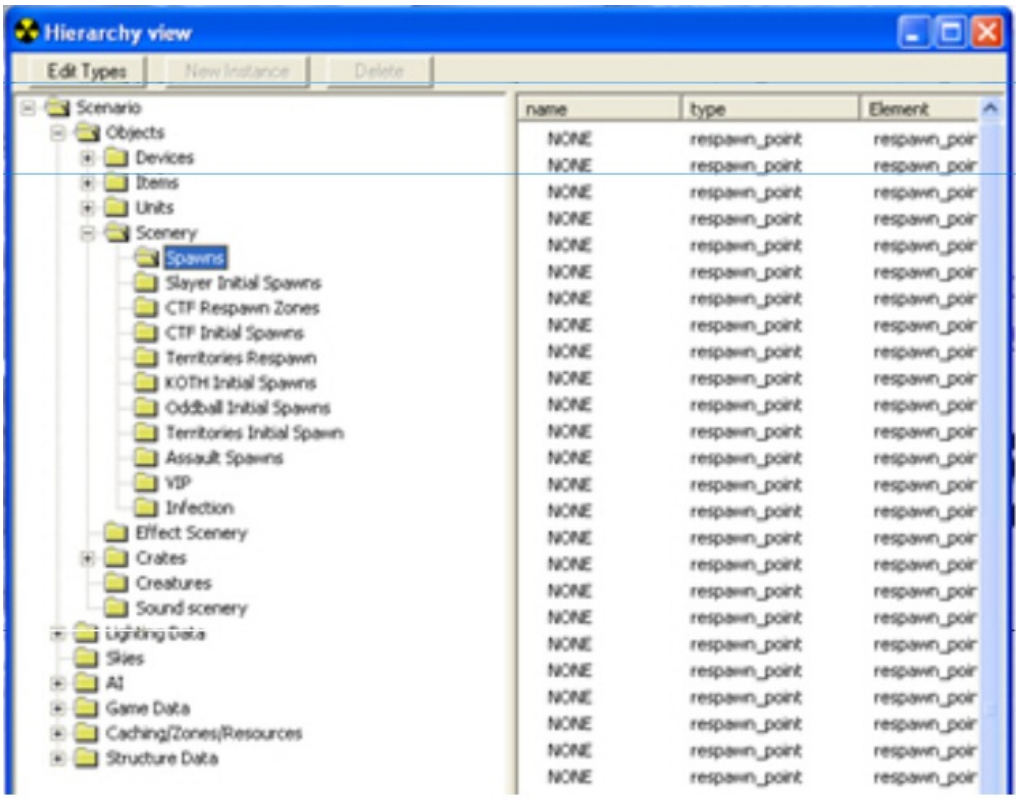


Figure 2 - Respawn Points

Figure 2 (above) shows subfolders in Scenery such as Slayer Initial Spawns, CTF Initial Spawns, etc. These are present just for organizational purposes right now even though they're not used. However, they will be used as you add more things to your map.

To make a subfolder right-click on the Scenery folder in the Hierarchy view and create a new folder.

A quick note about respawn points

Respawn points are where the player will respawn in the game and it's a subtle art to place them. A good rule of thumb is to try to respawn players in safe areas with their backs against a wall and facing towards a recognizable object or location in the level. These work across every game type. Place a few of these down to get a feel for it.

Initial Spawn Points

Every game engine looks for Initial Spawn Points when a match first starts, and this is where the first player starts on his team. They are broken up by game engine, so there are initial spawn points for CTF, Slayer, Assault, and every other engine in the game. For full map setup, you'll need to do this for every engine, but start with CTF so you can get a good feel for it.

Once again you need to add the objects into the palette. Go to your Hierarchy view, click Edit Types, select Object class scenery, and this time navigate to objects\multi\ctf\ctf_initial_spawn_point.scenery.

Once you place one of these down, you need to set the team designation. To do this, highlight your ctf_inital_spawn_point, and then go into the Properties palette. Almost all the way at the bottom is a drop-down list called owner team, and this is where you set whether this is the defending initial spawn or the attacking initial spawn. You only need to place one of these per team, because if you set up the map correctly, it will spawn the first person on the team at this point, and then try to spawn everyone else on the team near that person.

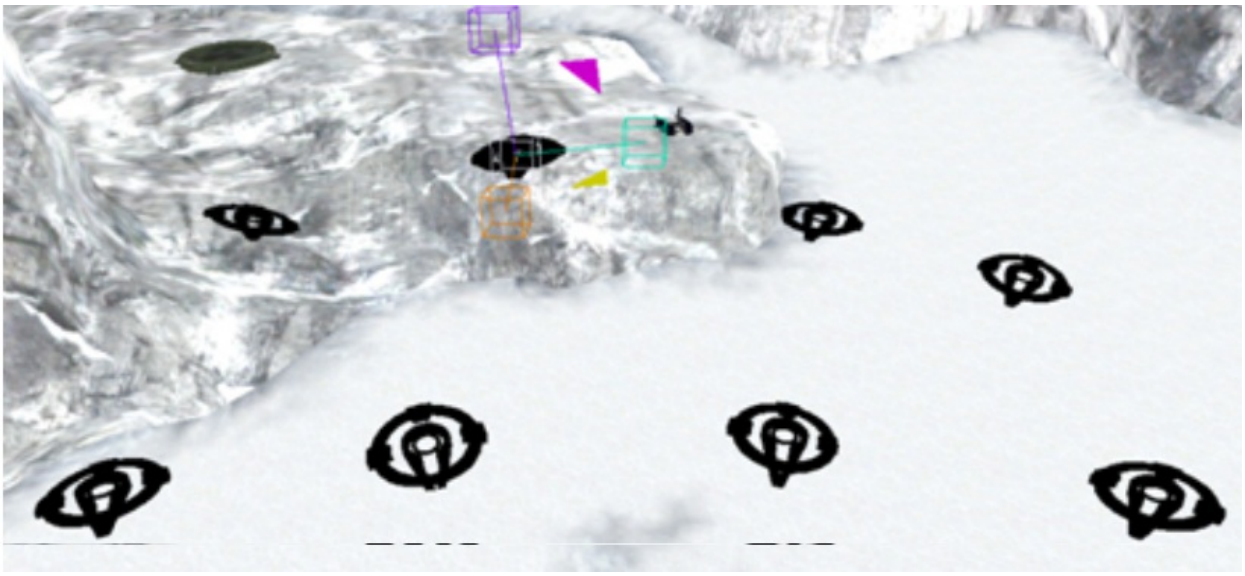


Figure 3 - Initial Spawn Points

Figure 3 (above) shows a highlighted point. This is the ctf defender initial spawn point. Notice how all the other respawn points are clustered around it. That will ensure that all your teammates spawn together.

Do the same thing for the attacker ctf initial spawn point, so you have one for each team.

Next we'll move onto placing the flag spawn and return points.

Game Engine Crates

You've placed the spawn points, but you still need the flag to show up in a CTF game.

1. Go to the Hierarchy view andd click Edit Types.
2. Switch your Object class to crate.
3. Navigate to objects\multi\ctf\ctf_flag_spawn_point.crate and objects\multi\ctf\ctf_flag_return_area.crate and add them both. Place them in a similar way to how you placed the respawn points, but this time since they are crates you'll need to use the Crates subdirectory, which is located below Scenery in the Hierarchy view.
4. Place one of each crate for each team.
5. Set the owner team in the Properties palette of each crate.
6. Place both the defender's ctf_flag_spawn_point.crate and ctf_flag_return_area.crate close to their initial spawn points, and preferably in the direction they are facing (also doing the same for the attacker). This helps players when they spawn so they immediately see where their flag is and what they have to defend.

Respawn Zones

Respawning zones are probably one of the more nuanced and difficult to manage things of the MP map setup. There are few rules that apply to every map (because they are all different), but you need to always be aware that wherever you set them up, that is where the player on that team will spawn. The fewer you place on the map the better, and in some cases bigger is better. Oftentimes covering one side of the map with a huge respawn zone is the best way to tackle them, especially on symmetric maps. If you ensure that the player will spawn where they are supposed to (e.g. not in the flag room in a CTF game), then you're on the right track.

Here's how to place them:

1. Go into the Hierarchy view, click Edit Types.
2. Ensure your Object class is set to scenery.
3. Navigate to objects\multi\ctf\ctf_respawn_zone.scenery and add it. You'll also see the flag_away and flag_at_home respawn zones as well. Don't worry about them for now as they are a little more complicated

and we only try to use them in maps when we really need to.

4. Place these the same way you place respawn points (since they are also scenery objects).

Once you've placed a node down, set the type to ctf_respawn_zone. They look like a little spike that sticks in the ground. Placing these down is just one part of the process, you also have to designate their shape. To do this, with your respawn zone highlighted, go into the Properties palette and scroll all the way down to the bottom to where you see boundary shape. If set to unused, it will default to a cylinder, but you can also set it to a box (we don't use sphere).

Underneath there are four numbers that can be changed to make the respawn zone the size you need it to be (the size is in world units (WU)). Look for boundary width or radius and that's where you start.

- Radius— the radius of the cylinder
- Box length— only used when the boundary shape is a box
- Positive height— how high up the volume goes
- Negative height— how far down it goes (you don't need to use negative numbers here)

If you want to see a visual representation of the zone, in your Tool window just check the flag for Draw transparent geometry. You will see respawn zones as a white wireframe. They will update in real time as you change the numbers. If you are not seeing the wireframes then make sure you have mp_debug_goal_object_boundary_geometry 1 in your editor_init.txt.

Territories

Some game engines use specific crate setups unlike any other. In Territories, for example, you have to set up the map for both Territories (Asymmetric) and Land Grab (Symmetric). To set up a map for Territories, first you have to add the territory flag into your map.

Using the same procedure as before, and adding crates this time, navigate to objects\multi\territories\territory_static.crate and add it.

Since you are basically setting the map up twice, we're going to learn how to set these up so they only show up only in one game type. First let's set up the map for Territories, which is the asymmetric variant.

1. Place a territory_static.crate down.
2. In the Properties palette scroll down to the MULTIPLAYER DATA section and you'll see a drop-down called game engine symmetric placement (see Figure 4).
3. Set game engine symmetric placement to asymmetric for Territories and symmetric for Land Grab.



Figure 4 - Placing Territories

The next thing that we haven't gone over yet, and is integral to Territories, is the idea of spawn ordering. Notice a

little further down in the Properties palette you'll see a box marked spawn order— this is how you number your territories. You'll always start your first numbering with 0, and then increase from there. So, in a five flag territories setup, you would have 0, 1, 2, 3, and 4 for your spawn ordering. When you're playing a game of Territories your flag that is spawn order 0 will be Territory 1 in game.

Another part of Territory setup is setting the boundaries of the capture volume. By default, there is a 3 wu radius cylinder that the player has to be in to cap a territory. After much playtesting, we realized that in most cases this too small in open areas. So, it's a good idea in most cases to increase the size of the capture volume. You do this the same way you set up respawn zone boundaries. You want to make sure the boundary positive height is almost 2 wu, so players don't get out of the volume when they are jumping and capturing. The actual radius size is mostly personal preference and map placement dependant, but don't be too stingy. If you check the Draw transparent geometry flag in the Tool window you can see your territory volumes. They will be a blue-green colored wireframe. If you are not seeing the wireframes then make sure you have mp_debug_goal_object_boundary_geometry 1 in your editor_init.txt.

A quick note about the name text field in the Properties palette; you'll notice that we have almost everything named in our maps. This becomes very important as your maps get more and more set up. If you get into a good habit of naming things early on, you won't be confused when you have to come back and fix or tweak things.

Destination Zones

Both VIP and Juggernaut use game engine crates we call Destination Zones, they are essentially waypoints on the map that the user has to get to. There is one for each game engine, but let's go over the VIP version first. Remember, what is talked about here applies to the Juggernaut version as well. They are both set up the same way, and the only thing that differs is where you place them on the map.

First you have to bring them into your map by navigating to objects\multi\vip\vip_destination_static.crate. Place one on your map and then in the Properties palette scroll down to spawn order again. Set up the spawn order like you did for Territories, starting with 0 as your first destination zone and increasing from there. Since these are used in Escort, it's important for the first destination zone to be far away from the defending team's initial spawn.

For Juggernaut your placement can be a little looser.

Hills

For King of the Hill, we're going to use objects with volumes again, just like the respawn zones. Add a hill marker by navigating to objects\multi\koth\koth_hill_static.crate. It uses the same model as the respawn zones, so don't worry about that. Since the hills move and can be in different locations, you'll need to use the spawn order box in the Properties palette. Once again, use 0 as the first hill you want to show up and then increase the number for subsequent hills.

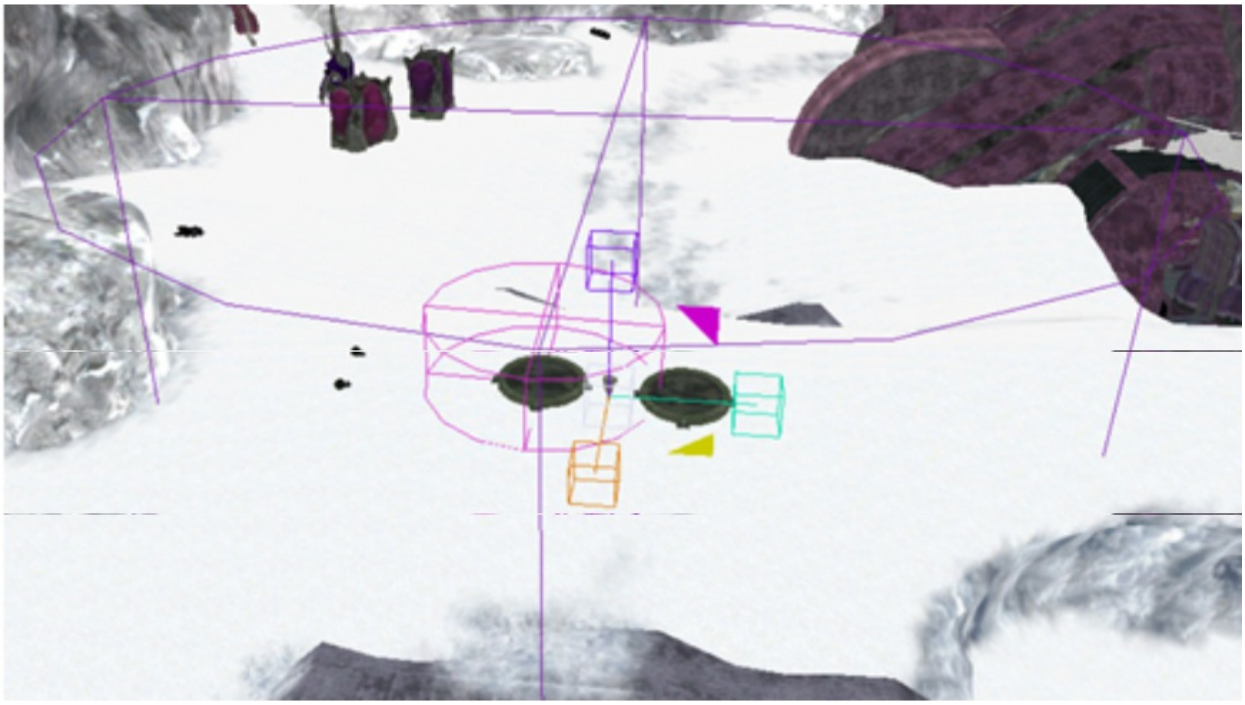


Figure 5 - Placing Hills

With hills, we can choose between two shapes, cylinders or boxes. You set this under boundary shape in the Properties palette. When you have chosen the proper shape for your hill, you can use the boundary width or radius, boundary box length (just for boxes), boundary positive height, and boundary negative height to adjust its size. Also remember that if you check the Draw transparent geometry flag in the Tool window, you can see what the hill looks like. Hill boundaries are purple wireframes in Sapien. If you are not seeing the wireframes then make sure you have "mp_debug_goal_object_boundary_geometry 1" in your editor_init.txt.

Oddball

The oddball spawn points are the locations you place for the Oddball to spawn. To place them in your map add a crate as usual and navigate to objects\multi\oddball\oddball_ball_spawn_point.crate. Like territories, hills, and destination zones you have to set the spawn order. A good rule is to always place the first oddball spawner in a neutral position on the map that both teams have to work equally hard to get to. In other words, having it in a central location that is approximately the same distance from each team's initial spawn point is a good way to think of placement for the first oddball spawner.

The Numbers

Those are the main ingredients for setting up the maps from a spawning and game engine standpoint. Each game engine has its own set of Initial Spawn Points, Respawn Zones, and game engine crates (VIP Destination Zone, Oddball Spawn Point). Setting things up for each game engine is something that becomes second nature after a bit, but making it fun is done through careful thought and lots of playtesting.

For reference on what a full map setup looks like, you could look at either Chill (as a symmetric map) or High Ground (for asymmetric). Listed below are the approximate numbers of objects per game engine, you may need to add more depending on the map and game engine:

Slayer

- 4× Slayer Initial Spawn point scenery objects (Defender, Attacker, 3rd and 4th Party)

Oddball

- 4× Oddball Initial Spawn point scenery objects (Defender, Attacker, 3rd and 4th Party)
- 5× Oddball Spawn Point crate (First one should be in a neutral location)

King of the Hill

- 4× KOTH Initial Spawn point scenery objects (Defender, Attacker, 3rd, and 4th Party)
- 3× KOTH Static crate objects (Set the size the same way you did the Respawn zones)

Capture the Flag

- 2× CTF Initial Spawn point scenery objects (Defender and Attacker)
- 2× CTF Respawn zone scenery objects (Defender and Attacker)
- 2× CTF Flag Spawn Point crate (Defender and Attacker)
- 2× CTF Flag Return Point crate (Defender and Attacker)

Assault

- 2× Assault Initial Spawn point scenery objects (Defender and Attacker)
- 2× Assault Respawn zone scenery objects (Defender and Attacker)
- 2× Assault Flag Spawn Point crate (Defender and Attacker)
- 2× Assault Flag Return Point crate (Defender and Attacker)

Territories

- 4× Territories Initial Spawn point scenery objects (Defender, Attacker, 3rd and 4th Party)
- 2× Territories Respawn zone scenery objects (Defender and Attacker)
- 10× Territory Static crates (5 of them symmetric and 5 asymmetric for Territories and Land Grab respectively)

Juggernaut

- 5× Juggernaut Destination Zones

Infection

- 2× Infection Initial Spawn point scenery objects (Defender and Attacker)

VIP

- 4× VIP Initial Spawn Point scenery objects (Defender, Attacker, 3rd and 4th Party)
- 5× VIP Destination Static crate (First one should be away from where the defending team spawns)

Weapons

Now comes the fun part, where we get to set up weapons on the map. At this point we're going to finally branch out into other resource tags than just scenery. To place weapons you'll need to make sure you have the weapons resource tag checked out (levels\multi\snowbound\resources\snowbound.scenery_weapons_resource). This is in the same folder that the scenery resource tag was in. Once you have that checked out, you can add weapons to the map the same way you always add things:

In the Hierarchy view go to Objects -> Items -> Weapons, and then click on Edit Types. In the dialog window, make sure weapons is the object class and then navigate to objects\weapons\rifle\battle_rifle\battle_rifle.weapon to place a Battle Rifle. Grab the other weapons you think you want to place on this map the same way.

Once you've collected a little arsenal in your palette you can start placing them around the level. Place them like any other object, and once you're happy with how it is sitting make sure you hit the Lock Placement to Object button located at the top of the Properties palette. This basically tells the game to keep the exact position for your delicately placed weapon. It will also respawn in that place.



Figure 6 - Weapon Selection

Placing a weapon is just the first step, however, and now we get our hands dirty with quotas and respawn times.

Each weapon can be set with a quota maximum, which can be found towards the bottom of the Properties palette. This tells the game how many of that type of weapon can be on the map at any given moment. So, for something like the Mauler it would be OK to have 6 of them on the map, you can set the quota maximum to 6. Obviously, each weapon is different, and you want to be careful with more powerful weapons like the Spartan Laser, Sniper, Shotgun, Sword, and Hammer; these you want to give low max quotas like 2 or 3 depending on the map. It's also very important that each separate instance of the same weapon have the same max quota. So, all your Maulers should be set to 6.

We can also set individual spawn timers for each weapon. These timers work in conjunction with the quotas, to make sure there are always weapons on the map when you want them to be. To set these find the spawn time text box in the lower part of the Properties palette, its right under the quotas. Common weapons like SMGs, and Spike Rifles usually have a low respawn time (~15 seconds), but more powerful weapons should have longer respawn times. The Sniper or Shotgun, for instance, can have anywhere from 90 to 180 second respawn depending on the map. These are all preference and how you feel the map to play, but if you want to get a good feeling for how we set up the existing maps, check out any of them for reference.

On every weapon you place make sure the flag does accelerate (moves due to) is checked. This allows weapons to be pushed around by explosions.

IMPORTANT

Ensure the following weapons are always in your map palette. They don't have to be placed, the game just needs to know they are in memory. These are the base weapons that are available in Player Traits, and they are mandatory for the game to function as designed:

- energy_blade
- magnum
- needler
- plasma_pistol
- assault_rifle
- battle_rifle
- shotgun
- smg
- sniper_rifle
- rocket_launcher
- brute_shot
- gravity_hammer
- spartan_laser

Equipment and Grenades

Placing equipment and grenades is just like placing weapons, but you use a different resource tag. Both grenades and equipment fall under the Equipment category. Make sure you have the equipment resource checked out (levels\multi\snowbound\resources\snowbound.scenario_equipment_resource), and then start populating. In the Hierarchy view go to Objects -> Items -> Equipment, and then click on Edit Types. Make sure the object class is set to equipment, and then add grenades and whatever equipment pieces you want for your map.

One big rule we've been using in our MP maps is that we are only using two grenade types per map; so it's always frags and then we choose between either brute or plasmas as our secondary. Once you have all the grenades and equipment you want to use, start placing them. We also need to set quotas and spawn times for them.

Grenades are unique in that there can be a lot of them on the map, and we want them to respawn quickly. So, by default set their quota maximum to 32 and leave their spawn time 0. When you leave one of the spawn time at 0, it will use the default set up in the main tag, and for grenades that default is 15 seconds.

Equipment should be treated more like power weapons on the quota and spawn times. A typical piece of equipment would allow for two on the map (quota maximum of 2) and a spawn time of 60 – 90 seconds. Once again these are personal preference and size of the map. So, some tweaking and playtesting will be required.

On every piece of equipment and grenade you place make sure the flag does accelerate (moves due to) is checked. This allows weapons to be pushed around by explosions.

Vehicles

Placing vehicles is just like placing everything else we've learned about, but once again we're using a different resource tag. In order to add and place vehicles you need the vehicle resource tag checked out (levels\multi\snowbound\resources\snowbound.scenario_vehicles_resource). In the Hierarchy view go to Objects -> Units -> Vehicles, and then click on Edit Types. Make sure the object class is set to vehicles and then add whichever vehicles are appropriate for your map.

Stand-alone turrets are technically vehicles so you add them here and not in the weapons category.

> [Note] turret.vehicle tags are in the objects\weapons directory rather than the objects\vehicles directory.

Just like weapons, you need to set a quota maximum and spawn times for your vehicles. These are dependent on the vehicle and the size of the map. For instance, it's fine to have a higher quota for Mongoose and have their respawn set to 10 seconds, but you wouldn't do that for a Scorpion or Banshee.

Make sure each vehicle has the create at rest flag unchecked. This is located under placement flags at the top of the Properties palette window and allows vehicles to spawn in and react to whatever is around them, instead of rigidly sitting there.

Objects

Once the level is a little further along, you'll want to start adding in flavor objects to fill out the world. You'll need the scenery resource checked out, and it's just like adding in the game object crates like the flag stand. You do this by once again clicking on Edit Types and setting the object class to crate. Click the Add button and browse to the objects you want to add.

To place an object, right click anywhere on the level to create a new node, then in the Properties palette at the very top is the type pull down menu. Choose from the existing objects you have added to place the one you want. Once you have it properly placed and it has come to rest, click the lock placement to object button at the top of the Properties palette. This will make sure it's location is exactly where you want it to be when the object

spawns and respawns into the level.

Once you have finished setting up all the crates you want, the last thing you'll have to do is count the total number of each one, and set that number as its max quota in the Properties palette. The easiest way to do this is to look at the objects in the Hierarchy view and sort them by type. Do this by clicking the type button at the top of the pane in the Hierarchy view. Then Shift-click all the objects of the same type, count them up, and while they are all selected go into the Properties palette and set all of their quota maximum to that number. So, if you have 8 crates, make sure each individual one has a quota maximum of 8. This will stop them from respawning in weird ways.

Trigger Volumes

One of the final things you will set up in a map is a kill volume. Most maps need this in some way or another, and it's up to you to decide where they go. First of all, to set one up you need to check out the trigger volume resource (levels\multi\snowbound\resources\snowbound.scenario_trigger_volumes_resource). Once you have the resource checked out, in the Hierarchy view open the Game Data section. Under the game data heading you'll see a Trigger Volumes sub-heading. Click in that sub-heading and then go into the Game window. Find the area you want to place a kill volume and right-click the mouse to create a cube.

To make the trigger volume fill the area where you don't want people to be, you can highlight a side of the cube and drag it to make it bigger. You want to make sure the trigger volume actually extends beyond the bounds of your bsp so you don't take any chances with someone being able to skirt around it. Better to be safe than sorry.

So, now that you've finished making this cube, we need one final thing to make this into a death box. When you highlight your trigger volume, look in the Properties palette and in the name box type kill_box. The game automatically converts any trigger volume with the works "kill" into a death volume. So, you can name it kill_house, kill_town, or any other number of things, as long as it starts with kill in the name. That's it— you've set up a kill volume!

After the Kill Volume there is one final volume you have to place on every MP map. After the soft ceilings are in place, you'll need to place the "safe_zone" volume that kills MP players when they leave it. This is a last ditch way to keep people from going places they shouldn't. Think of it as the reverse of a kill volume, in that everything inside it is all fine and cozy, anything that leaves the box, dies. So, create a trigger volume that encompasses and is just outside the soft ceiling, and then name it "safe_zone". Make sure it's far enough out from the soft ceilings so you can't boost as the monitor in Forge and get outside and mysteriously die. After you do a thorough check, you're done with trigger volumes.

Forge Setup

Setting up the Forge palette for a level is something done exclusively in Guerilla. It is all done in the scenario tag, so check that out for your level. The quickest way to set these up is to open the tag in the alternate mode. So, hold down the Alt key on your keyboard while double-clicking on the scenario tag in Guerilla. Once the tag is open, scroll down to the group of items that start with mv in their name, the first of which is the mv vehicle palette.



Figure 7 - Scenario Tag in Guerilla

When you double-click on one of the cells, you will see the items that make up that list. In the case of the vehicle palette, it will show you which vehicles are in the forge palette for that level.

There are certain palette items that never change across all the maps; these include the equipment, teleporters, goals, and spawners. Since these are already set up on all the shipping Halo 3 maps, the easiest thing to do is just a simple copy/paste.

Here's how...

1. Double-click the mv equipment palette in Deadlock.
2. Drag-select all of the entries and hit ctrl - c to copy all of them.
3. Back out to the main tree view again.
4. Double-click the equipment palette in your level then click the plus symbol to add more entries. Since the equipment palette has 15 things, create 15 cells.
5. Drag-select the blank cells and hit ctrl - v to paste them all in.

After you have copied all the ones that never change from existing maps (equipment, teleporters, goals, and spawners), you are ready to fill in your own things. You will have to make sure your weapon palette has the following guns:

- energy_blade
- magnum
- needler
- plasma_pistol
- assault_rifle

- battle_rifle
- shotgun
- smg
- sniper_rifle
- rocket_launcher
- brute_shot
- gravity_hammer
- spartan_laser

Once you have those in there, you can fill out the rest with first, whatever is definitely placed on your map. Do you have the Beam Rifle? Make sure it's in your Forge palette. Once you have all the weapons that are placed on your map in there, then it's time to get creative. Want to make sure there are Sentinel Beams so people can play Ghostbusters in Customs? Make sure it's in your weapon palette.

The same sequence goes for the rest of them as well. First, make sure all your pre-placed vehicles or scenery objects are in your forge palettes, but after that fill it in with fun things. Make sure there are big objects like crates and shipping containers. Look at prices from shipping maps, and go crazy. Remember this is what people will be using long after they get sick of our Matchmaking. Want a tank in that map, even if the default setup doesn't call for one? Make sure it's in your vehicle palette so players can put it in there with Forge.

A Note About Memory

There's a finite amount of memory available for each map and each weapon and vehicle takes up its share. It's often the case that weapons, vehicles, and other objects have to be cut from the final version of the map in order to come in under memory restrictions. Each map is examined on a case-by-case basis so there is no hard and fast rule on how many vehicles or weapons can go into a particular map, but every map gets something cut—don't be upset if this happens to you!

Miscellaneous

There are plenty of little things that need to be done on all of the maps; game object reset height, physics volumes, etc. This is where you'll find all of them.

Game Object Reset Height: This is how you tell the game objects (flag, oddball, bomb, etc.) to reset if they fall below the play area. In Guerilla if you open the scenario tag in Alt mode, scroll down to the cell spawn data and double-click it. There is an option called game object reset height, and you set the Z height where the object should respawn. The easiest way to get this number is just by creating a dummy node in your map in Sapien and dragging it down to a point outside and below the play space. Check the Z height, and then put that number into the cell.

Physics Volume shapes: In every map, we need to put a set of shapes that can be used later if necessary. These crates would be used if some glitches are ever found and need to be plugged up at a later date for Matchmaking. The following objects need to be placed far outside the playable space on your map. They are located in tags\objects\multi and you need two of each in every map:

- box_l
- box_m
- box_xl
- box_xxl
- box_xxxl
- wall_l
- wall_m
- wall_xl

- wall_xxl
- wall_xxxl

Garbage Volumes: In any map that has an open area or bottomless pit, there needs to be a garbage collection volume. This allows the game to clean up things that are sitting down there, which helps performance. An example would be the bottom of the level in Guardian. Since many bodies, weapons, grenades, etcz. fall down there it needs to be routinely purged.

To do this, set up a Trigger volume and name it garbage. Then make sure your script for the level has the following in it:

```
(script continuous guardian_main

(add_recycling_volume garbage 0 5)

(sleep 300)

)
```

Obviously you need to make sure the name of your map is in place of "guardian" in the above script.

Using Portals

11/23/2022 • 4 minutes to read

In it's simplest form, Portals divide environments into smaller spaces (called "clusters") to make it easier for the game engine to handle the rendering and A.I. This article will show you how to set up and maximize portals. The goal is to be able to shut down as much stuff as possible - both for rendering and A.I.

Types of Portals

There are four types of portals which can be set in the material tag of the portal:

- 2-way — Allows the player to see through to the other side of the portal from either side.
- 1-way — See only the way the triangles are facing. (For example, the one-way glass in Snowbound.)
- door — Only see through the portal when the door is in the open state.
- no-way — Cannot see through the portal in either direction.

Merging

If two portal doors are close together, for example side-by-side, the system will try to merge them together into one portal. To avoid this, add a number after the +portal name. For example: +portal1, +portal2, +portal3 and so on.

Portals in Detail

Here's a simple environment with two rooms (see Figure 1):

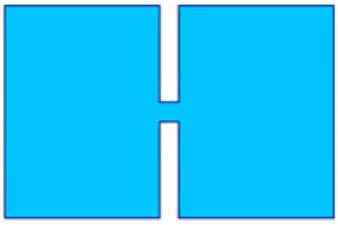


Figure 1 - Two Rooms

We can take that environment and subdivide it using a portal (see Figure 2):

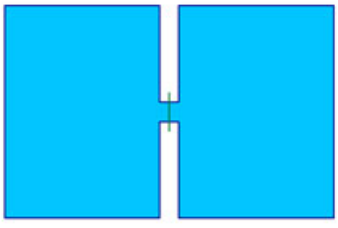


Figure 2 - Two Rooms with Portal

Which divides the environment into two clusters (see Figure 3):

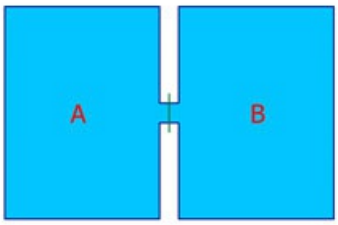


Figure 3 - Two Rooms with Clusters

Use the following commands in the console:

- switch_bsp 0— See the scenario without portals
- switch_bsp 1— See the scenario with portals
- visibility_debug_portals 1— See outlines of portals in the game
- debug_view 1— See through the walls (so you can see what's rendering) in the game

Simple Portals

If we add scenery objects (represented here by the colored dots in Figure 4)

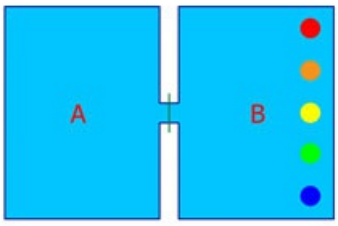


Figure 4 - Scenery Objects

The portals allow us to render only those objects that we can "see" (Figure 5):

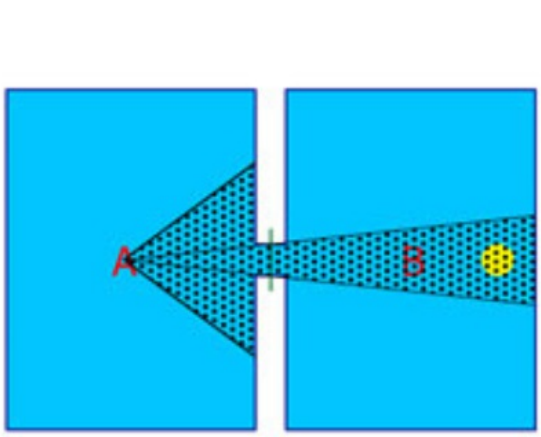




Figure 5 - Objects that can be seen

...start rendering objects that enter our field of view (see Figure 6):

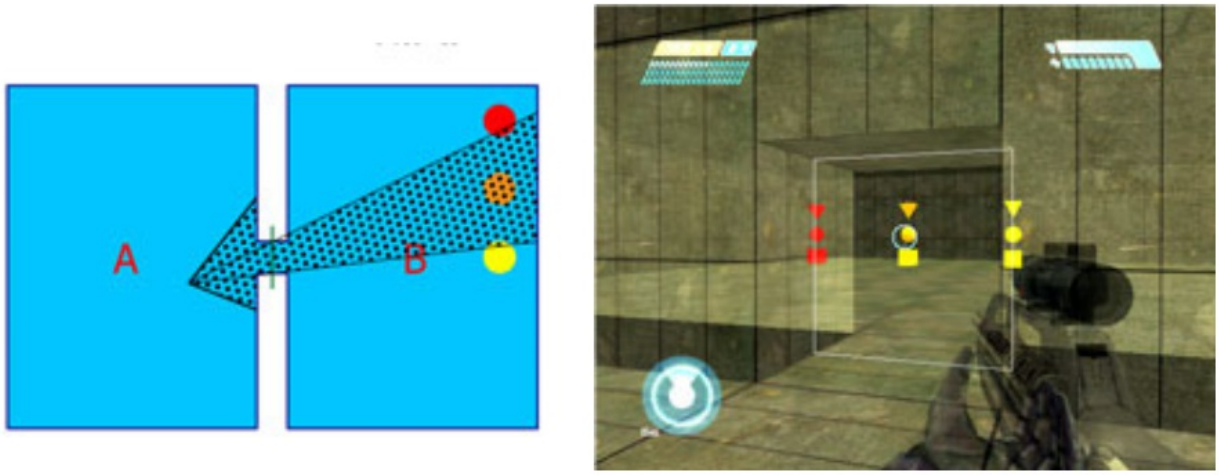


Figure 6 - Player Field of View

...and stop rendering objects that we can't (see Figure 7)

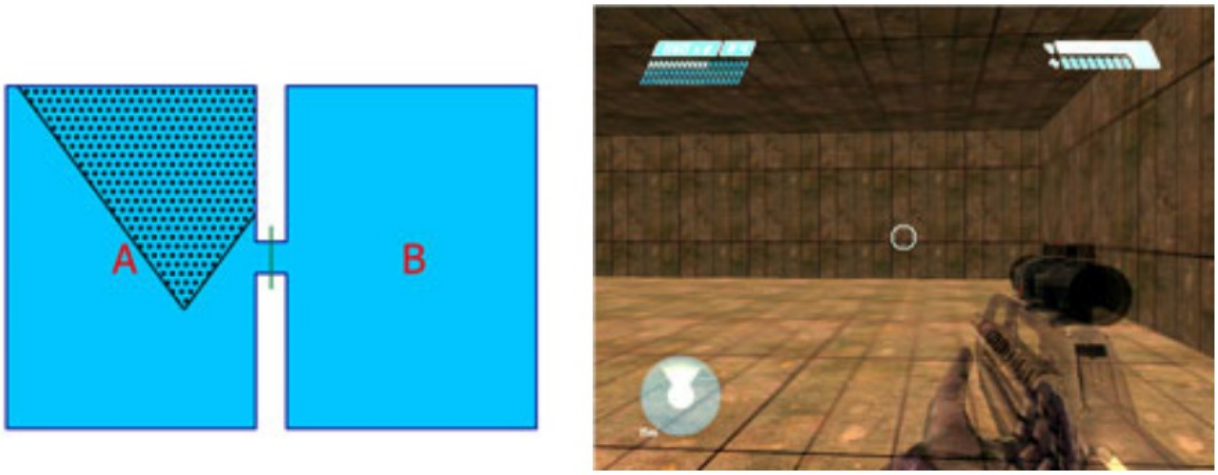


Figure 7 - Unseen Objects

Multiple Portals

Here's another environment, broken into eight clusters with portals (see Figure 8):

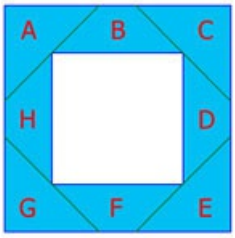


Figure 8 - Eight Clusters with Portals

If the player were standing in the top hallway on the letter B facing to the left they would be able to see into cluster A, but no other clusters. So we don't have to render the other six clusters (see Figure 9).

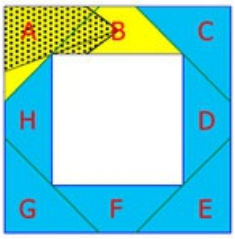


Figure 9 - Reduced Rendering while facing Left

If the player were facing to the right they would be able to see into cluster C, but no other clusters (see Figure 9):



Figure 10 - Reduced Rendering while facing Right

If the player walks around the environment in a clockwise direction (to stand on letter C), they can now see into clusters D and E:

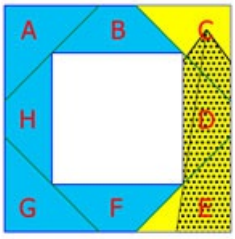


Figure 11 - Rendering if the Player stands in the C Cluster

...but other clusters "disappear" from view (see Figure 12):

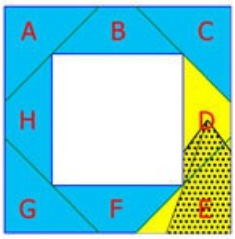


Figure 12 - Dissapearing Clusters

How It Works: Runtime Visibility Check

While the game is running, the engine checks to see if any of the cluster's immediate portals overlap with other portals. This is called the Runtime Visibility Check. If they do overlap, then that next cluster is rendered.

Figure 13 shows a hallway with four portals, breaking it into five clusters:

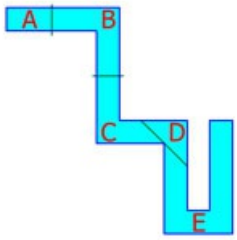


Figure 13 - Hallway with Four Portals

If the player were standing in cluster a, looking down the hall, it might look like Figure 14:

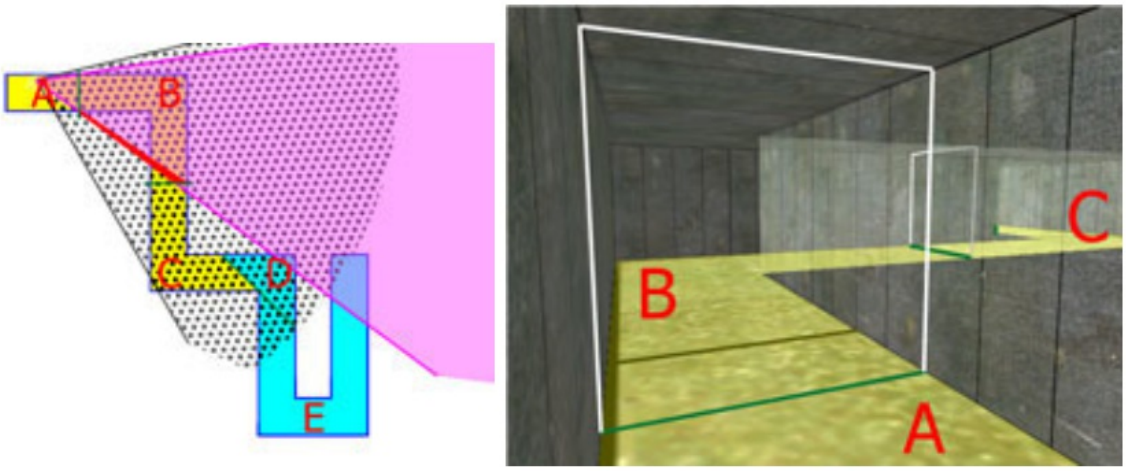


Figure 14 - Looking down the hallway

Remember that portals are not "hidden" by geometry. The Runtime Visibility Check only looks at portals, not geometry.

The player can "see" the portal between clusters B and C "through" the portal between clusters A and B.

Another situation is when the player is "behind" a portal as seen in Figure 15:

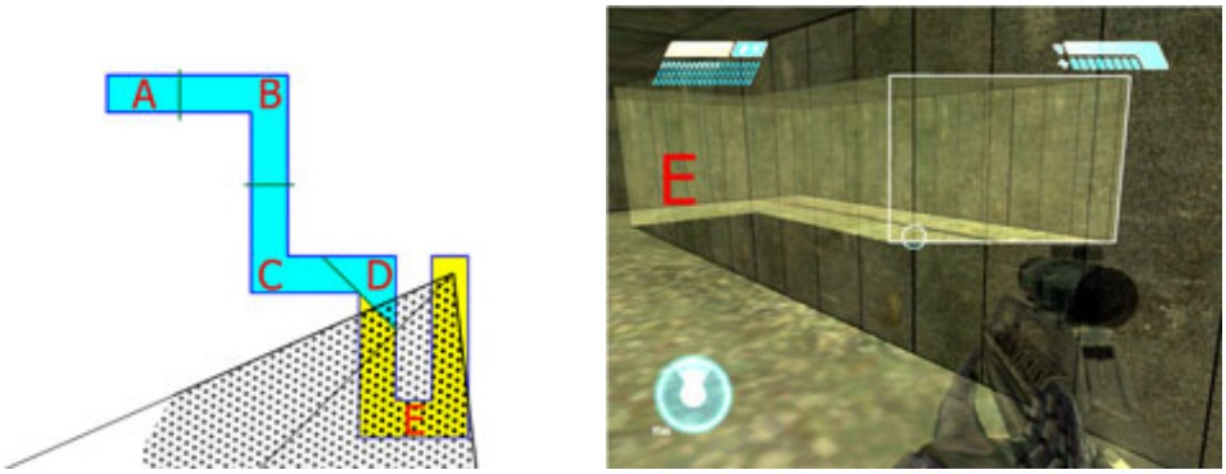


Figure 15 - Looking down the hallway

Potential Visibility Set

When an environment in imported, it generates a Potential Visibility Set for each cluster. This is a list off all possible clusters that can be seen from anywhere inside of that particular cluster.

The PVS is used for object activation. It is mainly used for the A.I., which can be shut off for inactive clusters.

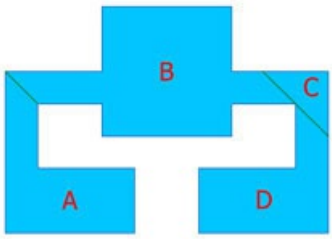


Figure 16 - Potential Visibilty Set Example

In this example, there are places in Cluster A, that can can see both Cluster B and Cluster C. There is nowhere in Cluster A where you can see Cluster D. So the PVS for Cluster A would include both Clusters B and C.

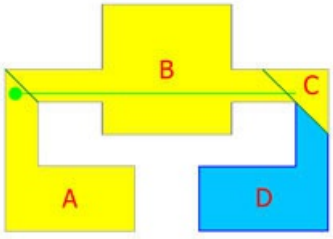


Figure 17 - Cluster D is not being rendered

Cluster B can see both Cluster A and Cluster C, but not Cluster D. Note that the portal arrangement on the right (separating B/C and C/D) works much better than the one on the left (separating B/A).

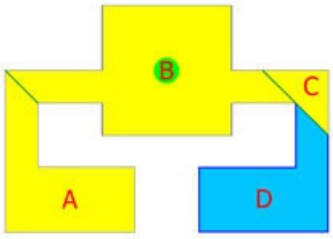


Figure 18 - Right vs. Left arrangement

Cluster D can only see Cluster C.

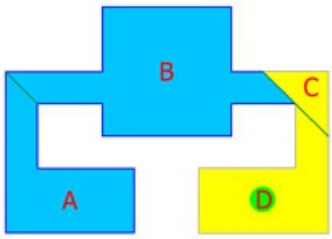
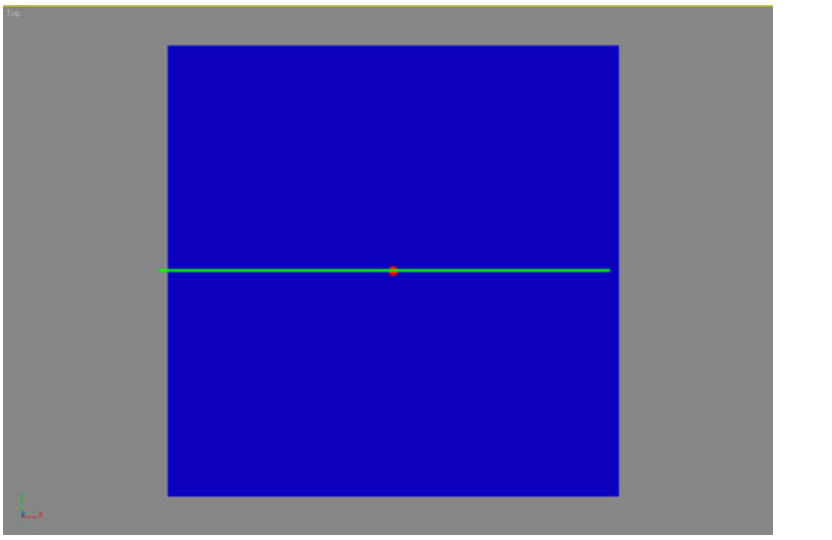


Figure 19 - Cluster D is not being rendered

Use debug_pvs 1 in the console to see the PVS (you must have debug_view turned on).

Common Errors

- **Exposed Portal Edge** — A common error is when the portal does not contact one edge of the environment (see Figure 20).



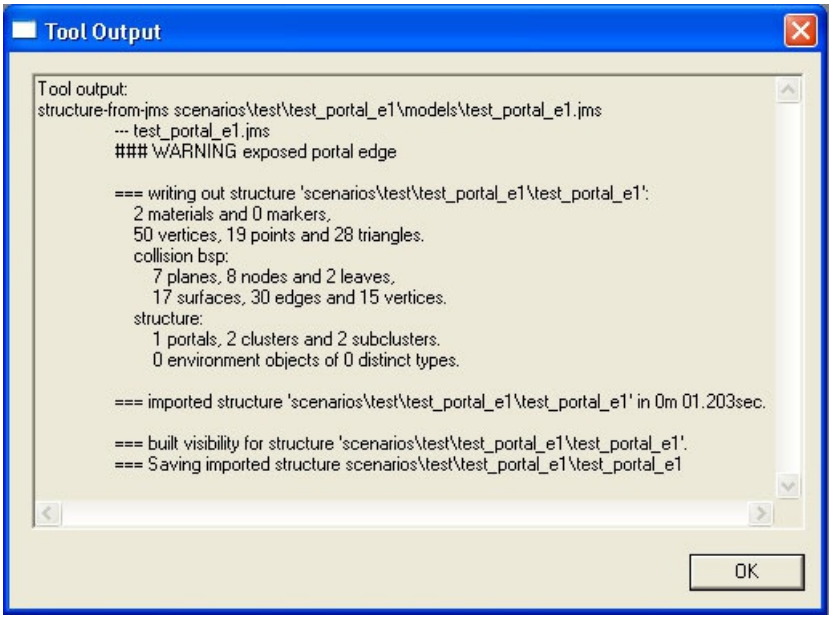
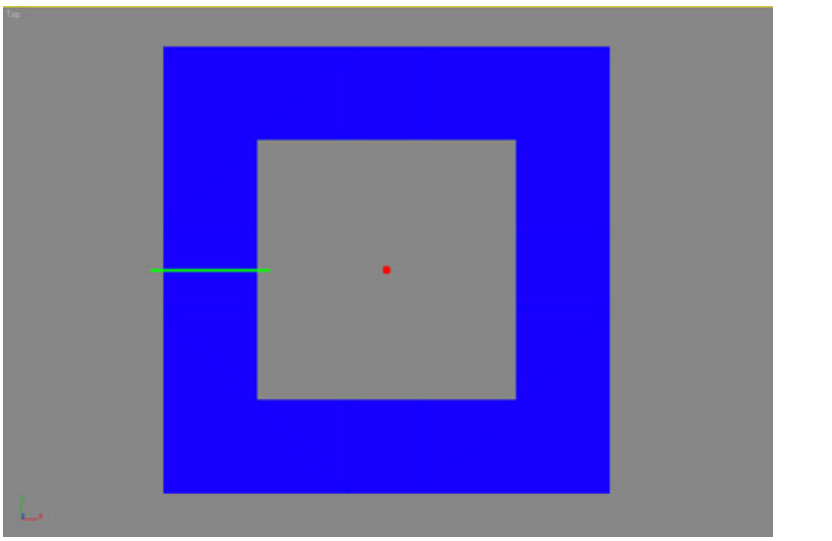


Figure 20 - Exposed Portal Edge Error

- **Portal Leakage** — A common error when the portal does not define two clusters.



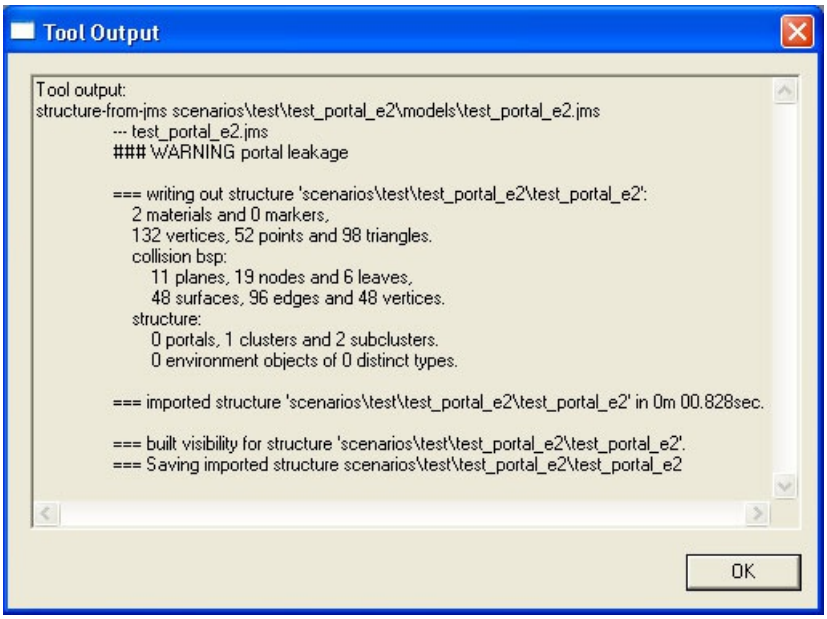
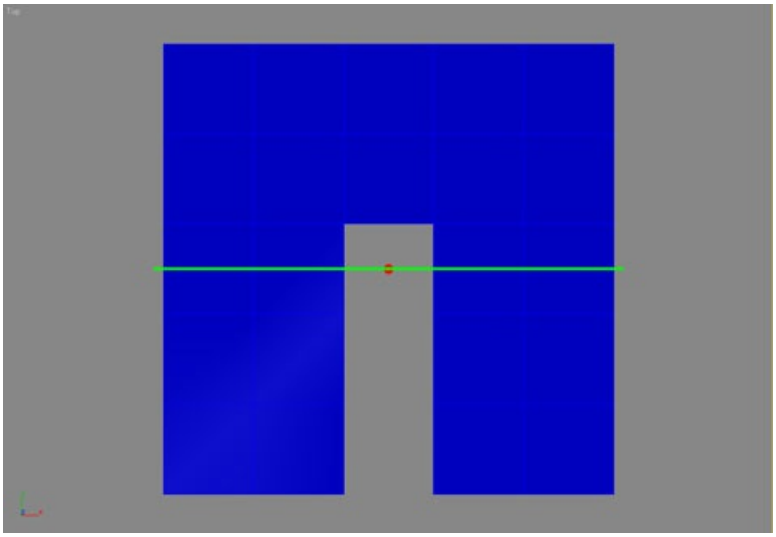


Figure 21 - Portal Leakage Error

- **Portal Couldn't Pick Cluster** — A common error when a single portal cannot be defined (see Figure 22).



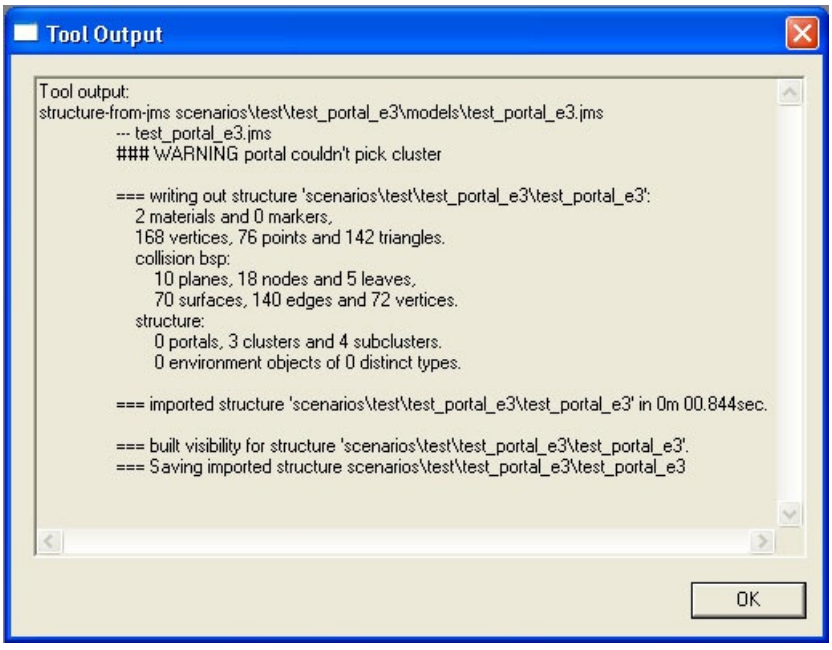


Figure 22 - Portal Couldn't Pick Cluster Error

Water Physics Volumes

11/23/2022 • 4 minutes to read

Overview

- Water physics volumes are placed in the structure design file.
- They don't have a material assigned to them, but require a special naming convention (see below).
- The shape can be anything as long as the surface is lined up with the water placed in the structure file.

Naming Convention

The naming convention for the water physics volume is:

~name— water physics volume, used to display water effects, allow objects to float in water.

#name— as a child of water physics volume, sets the direction of water flow inside the water physics volume. Water flows in (-) to (+) x direction.

Build a Water Physics Volume

Let's begin by building a water physics volume.

1. First you will need to find a map with Halo 3 water in it. This is usually a plane with material type Halo 3 water in the 3ds Max structure file, such as: \halox\atlas\data\levels\temp\water_simple\
2. Though we have the structure file for the water map, we also need a structure_design 3ds Max file, the same kind used by designers for placing soft surfaces, kill volumes, etc. Having this file will allow you to continually modify the design volumes in the map without exporting the entire structure over and over again. Go up one directory and create a new folder: \halox\atlas\data\levels\temp\water_simple\structure_design\
3. Create a new structure in 3ds Max (File —> New) and select Perspective View.
4. Go once again to the File menu and choose File —> XRef Scene.

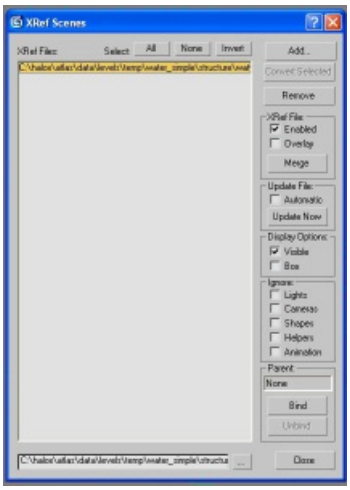


Figure 1 - XRef Scenes

5. Click the Add button in the top right corner of the window that pops up and navigate to \halox\atlas\data\levels\temp\water_simple\structure\water_simple.max. This should cause the structure of the map to appear in Max. You can now click Close.
6. Next, we must merge the water plane from the main structure file. Because we are using XRef, none of the water_simple map is actually contained in our new structure design file, everything is just visible for reference. To gain the parts of the map for making the water physics volume, we need to merge parts of the

structure file into our current one. Go to File —> Merge and navigate once again to the water_simple.max file, then click OK.

7. In the pop-up window, select both the ~water and #water_flow objects and click OK. Now the water portions of the map are available for editing in our structure_design file.

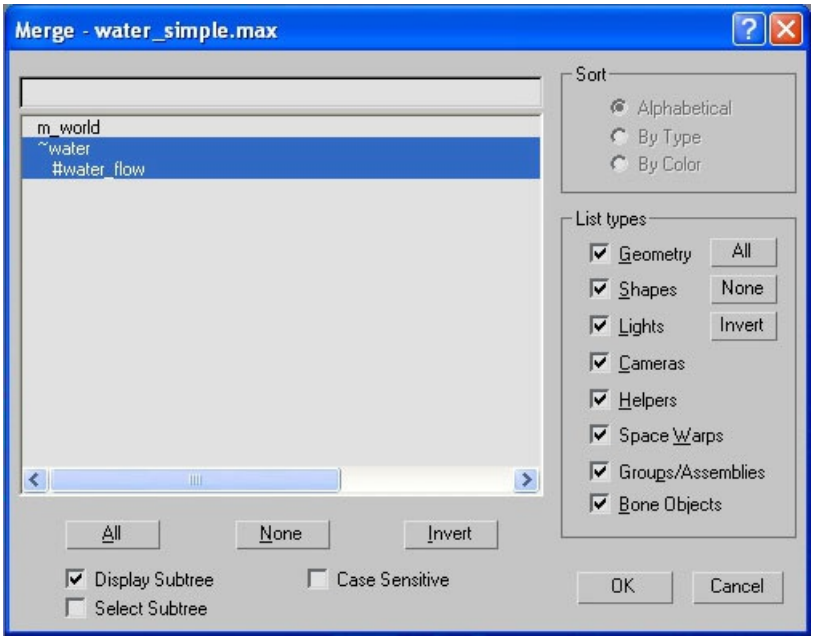


Figure 2 - Merging

> ![Note] If you didn't name the water with the ~ or # standards, change the names into ~<name> for bodies of water and #<name> for water flow objects.

8. At this point the structure_design file should be saved to your structure_design directory. Name the .max file the same as your normal structure file: \halox\atlas\data\levels\temp\water_simple\structure_design\water_simple.max
9. Normally you will need to make it a physics volume by turning the water plane into a box, but this has already been done in the water_simple map.

To do this yourself:

- 9a Select the all of the polygons in the top plane (ignore the bottom level).
- 9b Click the options box to the right of the Extrude button in the Edit Geometry section in the modifier tab.

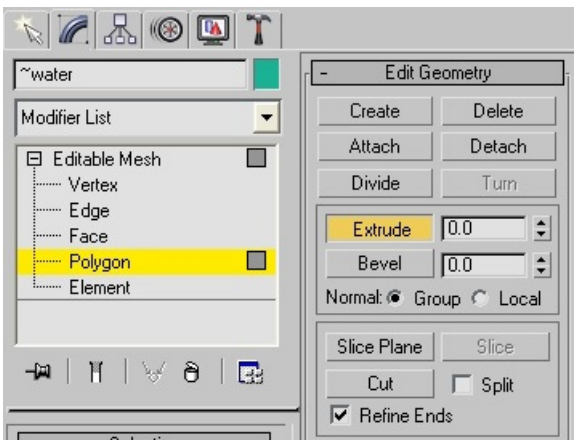


Figure 3 - Extrude Button

- 9c Click and drag the downwards arrow next to the button until the plane extends down below the bottom of

the map then click OK.

10. Your newly extruded shape now needs a cap. Select all of the polygons in your water plane, which should resemble a box without a top, and go to the Modifier List. Scroll down and select the option Cap Holes.



Figure 4 - Cap Holes menu item

11. Right-click the Cap Holes element above your Editable Poly element, and choose Collapse All. This will apply the cap to your water plane.

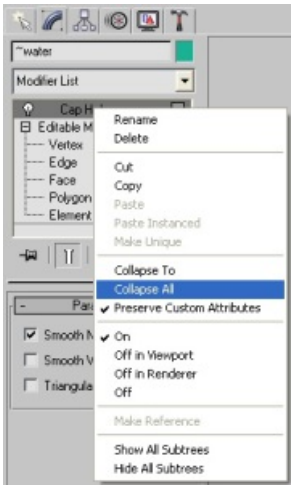


Figure 5 - Callapse All menu item

12. Finally, your water needs to have each poly facing outwards rather than inwards. Again select all the polygons in your water plane, go to your Edit Polygons modifier section and click the Flip button. All the polygons in your shape should now be facing outwards, making your water plane appear nice and solid.



Figure 6 - Flip button

13. Save the file and you're ready to export.

Congratulations! You've successfully built your first water physics volume. Now you need to export your file to an ASS file and import it into the game.

Export a Water Physics Volume

1. Choose Export from the File menu, and save it in the structure_design folder.
2. Launch Guerilla and run the tool command structure_design.
3. Browse to the ASS file that you just saved and run Tool on it.

Once it's successfully imported, hook it up to the scenario tag:

1. Launch Guerilla and open your scenario tag.
2. In the Structure BSPS section, click the ellipses (see Figure 9). The structure design file should be in the root of the tags folder for the level.

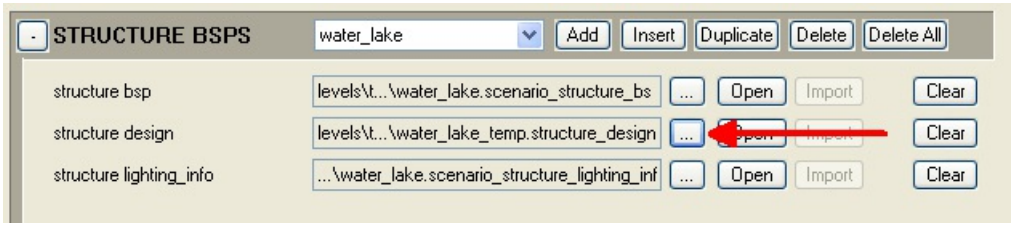


Figure 7 - Structure Design

3. Save the scenario

> ![Note] This only puts water physics on the level with no direction to the flow of water. This is good for ponds and standing water but not good for rivers. To add direction to water, see below.

Water Physics Direction

To add flow to your river, you will need to create an object and make it a child of your physics volume.

> ![Note] There is no way to set the flow speed, so only set the flow on moving bodies of water

1. Create a primitive object, like a simple box. The object can be placed wherever you like.
2. Name your object #waterdirection.
3. Make your object a child of your water physics volume.
4. Switch the view to local by selecting Local in the View pulldown near the top center of the 3ds Max window.
5. Rotate the object until the X axis is pointing in the direction you would like the water to flow. The water always flows in the +X axis direction.

Weapons

11/23/2022 • 2 minutes to read

Magazine Properties

This article contains information about the properties of the magazines for each weapon.

- **Weapon** — Name of the weapon
- **Rounds Total Initial** — The initial amount of ammunition the player receives when the weapon is picked up
- **Rounds Total Maximum** — The total amount of ammunition of this type that a player can hold at one time (in addition to the rounds loaded in the weapon)
- **Rounds Loaded Maximum** — The total number of rounds that can be loaded into the weapon at one time
- **Rounds Reloaded** — The number of rounds placed in the weapon each time the player reloads

Weapon Properties

WEAPON	ROUNDS TOTAL INITIAL	ROUNDS TOTAL MAXIMUM	ROUNDS LOADED MAXIMUM	ROUNDS RELOADED
Excavator	10	25	5	5
Magnum	32	48	8	8
Needler	75	105	15	15
Assault Rifle	96	192	32	32
Battle Rifle	108	144	36	36
Covenant Carbine	36	90	18	24
Shotgun	18	36	6	1
SMG	180	240	60	60
Sniper Rifle	12	24	4	4
Spike Rifle	180	240	60	60
Flak Cannon	20	30	5	5
Missile Launcher	2	4	1	1
Rocket Launcher	4	8	2	2
Spartan Laser	2	2	1	1
Brute Shot	18	24	6	6
Flamethrower	600	1200	100	100

Power Magazine Properties

WEAPON	POWERUP NAME	ROUNDS
Excavator	NA	NA
Magnum	pistol_ammo	24
Needler	needler_ammo	60
Assault Rifle	NA	NA
Battle Rifle	br_ammo	72
Covenant Carbine	NA	NA
Shotgun	shotgun_ammo	18
SMG	smg_ammo	120
Sniper Rifle	sniper_rifle_ammo	12
Spike Rifle	smg_ammo	120
Flak Cannon	NA	NA
Missile Launcher	rocket_launcher_ammo	1
Rocket Launcher	rocket_launcher_ammo	2
Spartan Laser	NA	NA
Brute Shot	NA	NA
Flamethrower	flamethrower_ammo	100

Scaling

Weapons are scaled in-game to accommodate both character size and game situation, so a weapon on the ground in multiplayer is scaled differently than when held in the hands of a Grunt.

This causes problems when authoring animation (both in-game and cinematic) as weapon size, weapon location, and hand placement are different in Maya than they will actually occur in-game. Some Maya files are currently being animated with arbitrary scaling on the weapon.

The solution to this problem is to add a scale factor to the next version of the weapons rig. This will automatically read the tag data and scale the weapon appropriately for the character. This will not adversely affect any existing animations, whether they have an arbitrary scaling to the weapons or not; however, there will be clean-up involved once weapons are their proper scale.

Weapon scaling info is held in **.weapon.item_scale_setting**

- single player ground
- multiplayer ground

- small unit armed
- small unit stowed
- medium unit armed
- medium unit stowed
- player unit armed
- player unit stowed
- large unit armed
- large unit stowed

Biped size info is held in **.biped.unit.item.item_owner_size**

- small, medium, player, or large biped

Additional Information

- Scaling affects the render model only, so the physics model must be scaled large enough to accommodate all of the different scales
- Atlas uses a smaller player character than Halo3: changes in weapons scale will need to happen in tags
- Omaha will be using a smaller player character than Halo3

Halo 3 Tools

11/23/2022 • 2 minutes to read

Information for general tools and scripts.

Device Machines

Device Machines are motorized, animated, objects placed in the game that either react to the player, or that the player can interact with.

Gameshell UI

Overview of the gameshell UI system. It contains information describing the core components, the tag structures, and useful debug commands and variables.

HaloScript Overview

HaloScript is the scripting language used within the game engine to make everything happen throughout the course of a mission.

Guerilla

Guerilla is the main interface for editing tags.

User Interface

Information on Guerilla's User Interface.

Edit Menu

Information on Guerilla's Edit Menu.

File Menu

Information on Guerilla's File Menu.

Filters

Information on Guerilla's Filters.

Source Control Menu

Information on Guerilla's Source Control Menu.

View Menu

Information on Guerilla's View Menu.

Window Menu

Information on Guerilla's Color Picker.

Color Picker

Information on Guerilla's Edit Menu.

Help Menu

Information on Guerilla's Help Menu.

Sapien

Sapien is the main interface for editing a scenario (MP and Campaign maps or Cinematic BSPs). This includes placing objects, decals, lighting, characters, vehicles, and setting up encounters with friendly and enemy AI.

Overview

A general overview of the Sapien tool.

Game Window

Information on Sapien's Game Window.

Hierarchy View

Information on Sapien's Hierarchy View.

Tool Window

Information on Sapien's Tool Window.

Asset Manipulation Gizmo

Information on Sapien's Asset Manipulation Gizmo.

Output Window

Information on Sapien's Output Window.

Placing Objects

Information on how to place objects in Sapien.

Placing Lights

Information on how to place lights in Sapien.

Properties Palette

Information on Sapien's Properties Palette.

Structure Painter

Information on Sapien's Structure Painter.

Device Machines and Controls

11/23/2022 • 17 minutes to read

Device Machines are motorized, animated, objects placed in the game that either react to the player, or that the player can interact with. Below, you can find detailed information on all of the properties of device machines, as well as step by step setup instructions to get you going quickly.

Things you should know

- The animation file for your device machine needs to be exported as a **.jmo**. None of the other animation formats will work.
- When creating the animation for your device machine, do not leave any static frames at the beginning or end! This causes problems later when adding sound effects.
- If you want the object's bounding sphere to move with it when it animates in game (and thus retain physics and collision), you need to animate the root node for the object (not any of the lower frame nodes).
- When you export your animation, it needs to be named **device position.jmo**. If the file is not named this way, it will not function!
- For position animations (which are what most device machine animations will be), our engine animates frame #0, but drops the last frame of the animation. For overlay animations, our engine doesn't play frame #0.

Step-by-step setup

- Create and import the geometry and animation files for your device machine. You will need render, collision, physics, and animations. Also, make sure you name the animation file **device position.jmo**. If you don't, the animation will not play.

When creating the animation for your device machine, do not leave groups of static frames at the beginning or end! This causes problems later when adding sound effects.

- eate any .shader tags for the materials you assigned to your model in Max. Re-import.
- Create a new .model tag. Link all of the other tags for your object to your new .model tag. Make sure you set the material types in your .model tag as well. Save it.
- In the Model tag block of the .model tag, click the Browse (...) button next to the **Animation** text box and link the .model_animation_graph tag for your device machine to your .model tag.
- Create a new .device_machine tag. Save it to a location of your choosing.
- Link your .device_machine tag to the .model tag for your object. This field is in the topmost section of your .device_machine tag (labeled Model) — simply click on the ... button and browse to the .model tag for your object.
- Scroll down to the bottom of the .device_machine tag to a block labeled Device. There are many different properties to check depending on the type of device machine you're trying to build (see below for detail on each property). To get your device machine up and running, the most important field to enter a number in is **Position Transition Time**. This is the amount of time it will take your device machine to go from closed to open (or off to on). Enter a number (in seconds) here now.
- Next, find the **Automatic Activation Radius** field. This is the distance (in world units) the player needs to be from the object for it to activate automatically. Think of this in the context of an automatic door — when the player gets close, it opens. Enter a number here now. Also, if your device is a door, and you don't want it to activate automatically, you'll need to check the Does not operate automatically flag in Sapien (See below

for more info).

- Now move down to the Machines tag block. For **Type**, select Gear. Even if you want to make a door, gear is the most useful for initial debugging of your machine because the gear type is always on - so you'll instantly know if your object is working. You can go back and reset the type to door at any time once you know the device itself (and particularly its animation) works.
- Open your scenario in Sapien.
- Add your new device machine to the Palette, and then place one (by right-clicking on the game window) in the scenario.
- Select the device you just placed (by clicking on its name in the Hierarchy View) and then select the type for it in the Properties Palette. You can find more information on properties set in Sapien in the Properties Set in Sapien section below.
- That should be everything you need to set up your basic device machine. Save and Xsync. If you already had your map running, you will need to reset the map before the device machine will begin to function. See below for more details on individual properties or flags you can set for your device.

Device Tag Block

- **Flags**
 - **Position Loops** — This flag allows the device to change its position to any value at will, but it will always choose the shortest circular path. When the position goes below zero or above one, it wraps around. For example, if you were at position 0.2 and wanted to to to 0.9, the shortest circular distance is to run backwards (.2, .1, 0, .9). This is different from the Gear machine type in that the Gear type always loops continuously in the direction of the animation you set up (it never goes backwards).
 - **Allow Interpolation** — Allows the device machine to interpolate animations (they don't do this automatically). Using this flag, you can set a series of positions (without animating them in 3ds Max) and the game engine will animate the device's movements. However, this will cause the animation to lag behind the device's actual position. For an example, think of the Scarab in Halo 2 (which didn't have animations, but was interpolated by the game engine). You should not use this for for things like doors or elevators.
- **Power Transition Time** — When a power group is set up for the device (with other devices, or with a device control), this controls the amount of time it takes for the device to go from its off state to its on state.
- **Power Acceleration Time**— The amount of time it takes for the speed of the power transition to accelerate from 0 to 1— or decelerate to 0— when powering up.
- **Position Transition Time**— The amount of time (in seconds) it takes for the device machine to go from a completely closed (off) to a completely open (on) state.
- **Position Acceleration Time**— The amount of time (in seconds) it takes for the device machine to reach it's top speed (or slow down to zero if it is in closing mode).
- **Depowered Position Transition Time** — The time it takes for the device to enter its depowered state (from a powered state). This only matters when the device is part of a power group (set up in Sapien) with other devices or device controls.
- **Depowered Position Acceleration Time** — The amount of time it takes for the device to accelerate from stopped to top speed on its way into the depowered state. This only matters when the device is part of a power group (set up in Sapien) with other devices or device controls.
- **Lightmap Flags**
 - **Don't use in Lightmap** — When this flag is checked, the device will not be considered when lightmaps are run. So, for example, if the device is covering an area (or blocking light), the lightmapper will run as if the device did not exist and the area will be lit by lights that might otherwise be blocked.
 - **Don't use in Lightprobe** — When this flag is checked, the lightprobes will not consider the way a device machine is affecting lighting in the area and, therefore, how the player is lit when standing on/around/near the device.

- **Open (up)**— The effect that is played while the device is opening (or moving up in the case of a lift).
- **Close (down)**— The effect that is played while the device is closing (or moving down in the case of a lift).
- **Opened**— The effect that plays while the device is opened/on.
- **Closed**— The effect that plays while the device is closed/off.
- **Depowered**— The effect that plays when power to the device is shut off.
- **Repowered**— The effect that plays when power to the device is turned on.
- **Delay Time**— The time (in seconds) between when the device is activated and when it actually begins to move.
- **Delay Effect**— The effect that plays during the time between when the device has been activated and when it actually begins to move.
- **Automatic Activation Radius**— The size of the radius (in world units) within which the player needs to stand to activate the device.

Machine tag block

- **Type**— Select the type of device machine you want to create from the drop down list:
 - Door — The device will behave as a door — opening and closing depending on the settings you choose. Doors are set to open automatically when the player comes within their activation radius unless they are part of a position group with a device control (and you set the Does Not Operate Automatically flag in Sapien).
 - Platform — A platform behaves the same as a door device except that it will not activate automatically. For platforms to function, they need to be part of a position group with a device control (a button, for example).
 - Gear— The device will work like a gear— constantly performing it's animations in a single direction— never going backwards and only having the ability (via setup in the tag) to speed up or slow down. In this state, the player doesn't need to turn it on or off, or activate it in any way.
- **Flags**
 - Pathfinding Obstacle — Checking this flag prevents AI from pathing onto the device— they will consider it an obstacle and path around it.
 - ... But not when Open — Checking this flag in conjunction with Pathfinding Obstacle will allow AI to path onto or through the device (or space created by the device) when it is open (on) but not when it is closed (off).
 - Elevator — Allows the device to stop and start at multiple points along the way from completely closed to completely open. For example, the silo elevator in 04b_floodlab (Halo 2). The stopping and starting can be accomplished using scripting (see Tyson or Rob for more info on how to do this). For the elevator to work, you also need to enter an Elevator Node below (in the Machine properties). However, contact Petar to see whether option is disabled in code (elevators may not work at the moment).
- **Door Open Time**— The amount of time (in seconds) a door will remain open before closing (unless it has an automatic activation radius which the player is standing within, or if the door is controlled by a device control— in either of those cases, this property is ignored).
- **Door Occlusion Bounds**— This flag activates and de-activates visibility portals within the distance of the door that you specify (in world units). This is used so that things don't render when the door is closed (or render if we need them to so that they don't pop in when the door is opened by the player).
- **Collision Response**— How the device responds when it collides with another object.
 - **Pause Until Crushed — The device will continue to its normal open or closed position, crushing anything in its path.
 - **Reverse Directions — The device will stop and reverse directions when it collides with an object.
- **Elevator Node** — Used in combination with the Elevator flag (see above) to create device machines which can stop at various points between their fully open or fully closed states.

- **Pathfinding Policy**— Sets the way AI will path on, through, or around the device.
 - Discs — Pathfinding for this device will be generated using a disc-type style.
 - Sectors — Pathfinding for this device will be generated using sectors.
 - Cut_Out — Pathfinding for this device will be generated using cut outs.
 - None — No set pathfinding policy. The engine will decide.

Properties Set in Sapien

The special properties for Device Machines in Sapien are found in the Properties Palette (after you've placed your device machine in the game window).

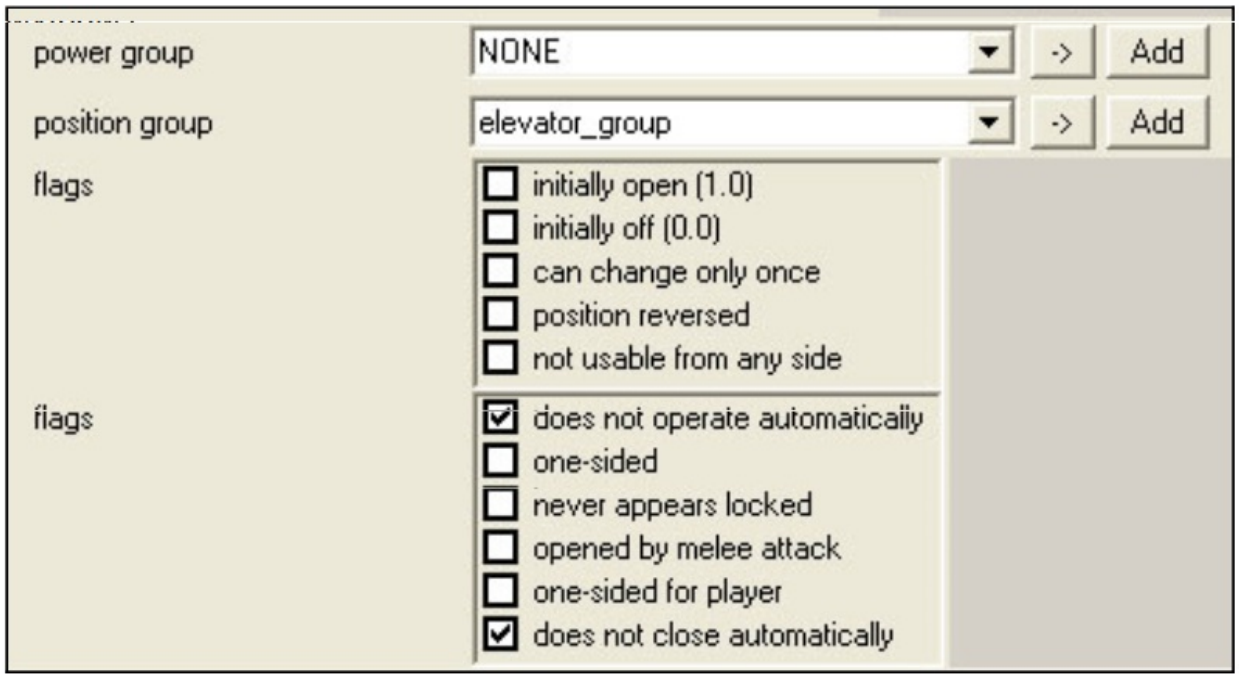


Figure 1 - Device Machine Properties Set in Sapien

- **Power Group**— Power groups are set up to link device machines and device controls together when the machine has a depowered and powered state.
- **Position Group**— Position groups are links between device controls and device machines. This group controls the animation/movement of the machine. If you want to set up any device machine which is controlled by a button (or any other device control), you must group it in a position group with a device control.
- **Flags**
 - Initially Open (1,0) — The device will appear in its open/on state when the game/level starts.
 - Initially Off (0,0) — The device will be in its depowered/deactivated state when the game/level starts.
 - Can Change Only Once — Allows the device machine to change position (open or close, for example) only one time. Once it has moved, it cannot be moved again.
 - Position Reversed — Reverses the position values for the device. So, a door that would normally start out in its open state would start out in its closed position when this flag is checked.
 - Not usable from any side: Prevents the player from being able to use the device.
- **Flags**
 - Does Not Operate Automatically — the device will not operate automatically. If this flag is set, the automatic activation radius property in Guerilla will be ignored, and you will need to set up a position group with a device control in order to operate the machine.
 - One-Sided — the device will only operate from one side. The side can be set as the positive x-axis of a marker named one_way or the x-axis of the door (or other device) itself if no marker exists.
 - Never Appears Locked — For door objects. The device will never report itself as locked to object

function callers. In other words, it will never appear locked to the player (appearance-wise).
 - Opened By Melee Attack — when this flag is set, the player will need to melee the device machine for it to enter its opened state.
 - One-sided For Player — The device will only be accessible to the player from one side. AI can access it from either side. See "One-Sided" flag description for more information.
 - Does Not Close Automatically — prevents a device machine from closing automatically. You will need to set up a position group with a device control (such as a button) to get the device to close.

Device Controls are used to activate or deactivate device machines.

Things You Should Know

- Device Controls need to be linked to Device Machines using a **Position Group** in Sapien. Position groups are set up so that the position animation will play. Controls and machines can also be linked in a **Power Group** in order to power on or off device machines (power groups will not play position animations).
- Device Controls can not have physics (although they can have collision), so they need to either be placed on a piece of geometry, or linked to an object to give the appearance of having physics.
- Device Controls can not have position animations. If you want to animate your device control, you need to attach it to an object using the objects_attach script command.

Setup Instructions

1. Create and import the geometry (render and collision) for your Device Control.
2. Create a new .model tag. Link your render and collision tags to it.
3. At some point, don't forget to create .shader tags for the materials you set in Max and then re-import your model.
4. Create a new **.device_control** tag. Link it to the .model tag you created previously. This field is in the topmost section of your .device_control tag (labeled Model) — simply click on the ... button and browse to the .model tag for your object.
5. Scroll down to the very bottom of your device_control tag to the block labeled **Control**.
6. In the Type drop down list, select **Toggle Switch**. You can always come back and change this later if you're not intending to build a toggled device.
7. In the Triggers When drop down list, select **Touched By Player**.
8. In the Action String text box, you can enter any text you'd like to appear when the player gets near the object. This is the text that appears on-screen and tells the player what to do to activate the device. See below for detailed information on how this string works (and how to set it up with a button icon).
9. Now is a good time to save all the tags you just created. Please do so.
10. Open a scenario in Sapien (the one where you'd like to place this control— and a device machine if one is not already present).
11. Add your new device control to the Palette, and then place one (by right-clicking on the game window) in the scenario.
12. Select the device you just placed (by clicking on its name in the Hierarchy View) and then select the type for it in the Properties Palette.
13. In order for your device control to work with a device machine, you need to link them together in a **Position Group**. Click on the Device Groups folder in the Hierarchy View and choose **New Instance**. Choose a fitting name for the group.
14. Now go back to the Controls folder in the Hierarchy View. Select your Device Control in the Pane on the right. In the Properties Palette, assign your control to the new group you just created (Select the name of the group from the **Position Group** drop-down list).
15. If you haven't placed/created a device machine yet, do so now.

16. In Sapien, make sure your device machine has the **Does Not Operate Automatically** flag checked (otherwise it will ignore your new device control).
17. Assign your Device Machine to the same **Position Group** as the one your device control is assigned to.
18. Save and reset the map. You should now be able to activate/de-activate your device with the device control.

Control Tag Block

- **Type**
 - Toggle Switch— If the current position of the device (grouped with the control) is greater than 0.5, then the device is told to go to zero (off or closed). Otherwise, it is told to go to 1 (on or open). Switches the device to the opposite of its current state.
 - On Button— This sets the control so that whenever it's pressed, it tells the device to go to 1.0 (fully on).
 - Off Button— This sets the control so that whenever it's pressed, it tells the device to go to 0.0 (fully off).
 - Call Button — Whenever hit, this tells the target machine to go to the position listed in the control definition (this is the Call Value field - enter any number between 0 and 1).
- **Triggers When**
 - Touched by Player— The control will activate when the player presses and holds the "X" button when standing within the activation radius (getting the activation message).
 - Destroyed**— The control will only activate when it has been destroyed.
- **Call Value** — Any value between 0 and 1 (0 being off/closed, 1 being on/open). This value is used in conjunction with setting the control type as a Call Button.
- **Action String**— The string that will appear on-screen in the player's HUD telling them how to activate the device. You can type any string you want here, but if you want button icons (or other pictures) to appear, you have to place a special string in the HUD_messages (\main\data\UI\hud_messages) text file and then call the string here.
- **On**— The effect that plays while the control is in its on state.
- **Off**— The effect that plays while the control is in its off state.
- **deny**— The effect that plays when the player is denied activation of the control.

Properties set in Sapien

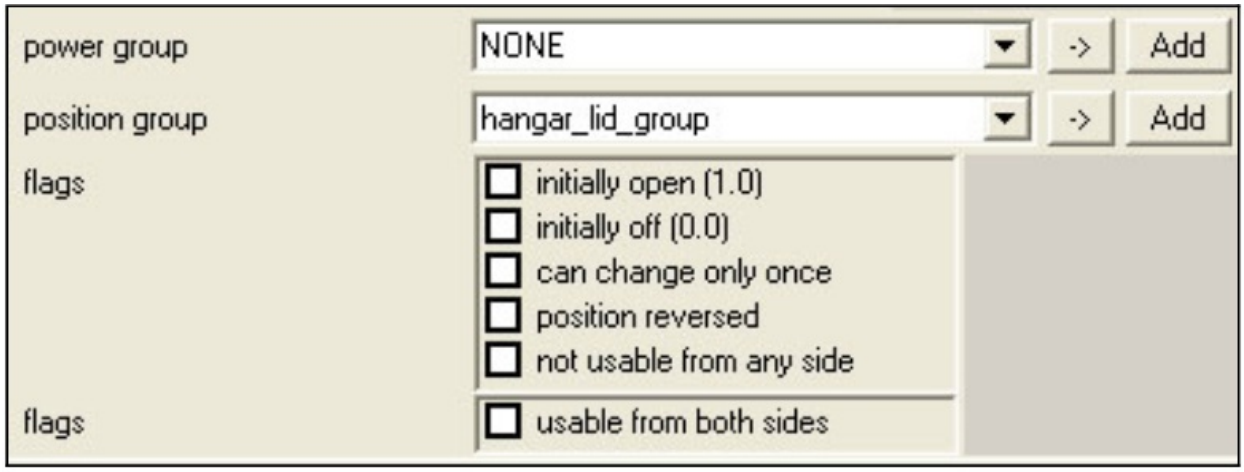


Figure 1 - Sapien Properties for Device Controls

- **Power Group**— Power groups are set up to link device machines and device controls together when the machine has a depowered and powered state.
- **Position Group**— Position groups are links between device controls and device machines. If you want to

use a device control (such as a button) to control a device machine, you need to make a new position group and add both the device control and the device machine to it.

- Flags
 - initially open — The control will appear in it's open/on state when the game/level starts.
 - initially off — The control will be in it's depowered/deactivated state when the game/level starts.
 - can change only once — Allows the control to be used only one time (thus preventing the device machine from being used more than once as well).
 - position reversed — Reverses the position values for the control. So, a button that would normally start out in its on state would start out in its off state when this flag is checked.
 - not usable from any side — This flag sets the control as unusable by the player (although the AI could still use it).
 - usable from both sides — The player will be able to access the control from any side.

Gameshell UI

11/23/2022 • 40 minutes to read

This document provides an overview of the Halo3 gameshell UI system. It contains information describing the core components, the tag structures, and useful debug commands and variables. It's intended for UI designers and developers who want a high-level overview of the system.

Core Components

The central structure is the *widget*. All gameshell UI is built by assembling one or more widgets in a hierarchy, with the widgets being responsible for rendering and running simple logic (responding to external messages such as controller inputs and generating other messages based on current state or inputs into the UI system).

Widgets

Widget is the term used since Halo1 to describe an element of the gameshell UI. There are various types of widgets, but they all share some common properties. The common attributes of a widget include:

- A name
- A type (or class)
- Linkage with other widgets in a hierarchy. All widgets can have a parent and any number of children. Widgets also have knowledge of their siblings
- The ability to render itself
- The ability to animate its render properties
- The ability to run logic at game update time

The widget's name is important for identifying a particular widget. In conjunction with the widget's type, it's used for locating a particular widget within a given hierarchy of widgets for things such as messaging (either with other widgets or with external game systems). Therefore, an important guideline to keep in mind when authoring UI is that if a widget is ever going to perform any interesting work, you should give it a unique name. In the Halo3 UI system, these names are string_ids.

The widget's type refers to what kind of widget it is. The widget's type determines what general properties and functionality it has. Types of widgets include: text widgets, bitmap widgets, model widgets, group widgets, list item widgets, list widgets and screen widgets. The following sections will drill down into more detail on these different types of widgets.

All widgets share a common set of animated rendering properties. Some of these properties are relevant only to specific types of widgets. These animatable attributes are:

- 2D rectangular boundary (in screen-space)
- Rotation applied to the rectangular boundary
- 2D scale applied to the rectangular boundary
- 32-bit color (including transparency)
- 3D positional offset applied to the rectangular bounds as well as the origins for rotation and scaling, if applicable (the animation of depth here can produce similar effects to the 2D scaling due to the perspective projection with which the UI is rendered in a simulated 3D space)
- 2D texture coordinate offsets (applicable only for bitmap widgets)
- Sprite sequence and frame (applicable only for bitmap widgets)
- Font (applicable only to text widgets)

- Render depth bias (not animated)— used to give rendering depth priority to widgets at the same three-dimensional depth

Text Widgets

A text widget is needed any time you want to render a block of text in the gameshell UI. In addition to the common widget properties, text widgets also have the following custom properties (these may or may not all be exposed in tags):

- Font
- Horizontal and vertical justification
- Drop shadow style (none, standard drop shadow, or full outline)
- 32-bit color and shadow color (including transparency)
- Scale
- Height adjustment
- Independent render and clipping bounds
- Configurable tab stops
- Configurable initial and paragraph indents
- Scroll amount
- Can pulsate
- Can have wrapping brackets applied to the contents when the widget has input focus

The contents of a text widget can come from one of three sources:

- The text string corresponding to the parent screen's string list tag entry with the same name as this text widget's **value identifier** field (this is the default behavior)
- The text string corresponding to the parent screen's string list tag entry with the same name as this text widget's parent list item widget label
- The text string corresponding to the parent screen's string list tag entry with the same name as the string_id value exported by this text widget's parent list item widget for the key value named by the text widget's **value identifier** field

The contents of text can be changed inline due to XML tags inserted in the text stream. Additionally, certain text properties such as color and font can also be changed inline via XML tags.

> **NOTE**
> Due to the use of XML parsing, care should be taken to avoid inserting raw < and > characters into text strings used by the gameshell UI.

In debugging builds, if the game is unable to determine the contents for a text widget, the text widget will display #unknown# as its contents field. This will enable you to quickly identify improperly configured text widgets.

Bitmap Widgets

A bitmap widget is needed any time you wish to display a bitmap in the gameshell UI. In addition to the common widget properties, bitmap widgets also have the following custom properties (some of which may or may not be exposed in tags):

- Tag reference to a bitmap tag. This is the tag from which sprite bitmaps will be pulled when rendering this widget as a sprite
- Tag reference to a shader tag (currently unused)
- Render blend mode. This is the color blending method used when rendering this widget as a sprite

- Initial sprite sequence. This is the initial sprite sequence used prior to any animation (the sprite sequence derived from animation and used when rendering uses this as the baseline value)
- Initial sprite frame. This is the initial sprite frame used prior to any animation (the sprite frame derived from animation and used when rendering uses this as the baseline value)

3D Model View Widgets

Model view widgets are not presently implemented. However, the plan is for them to consist of a group of models positioned relative to a local origin in space, along with a group of similarly positioned lights. This group of objects and lights will then be rendered into the widget bounding area.

Group Widgets

A group widget is a container for a collection of other widgets. Its primary purpose is to act simply as an organizational tool. In and of themselves, they do not render anything to the screen, however they are able to respond to inputs and perform simple logic. For example, they are able to trigger responses to common messages (this is described in more detail later in the section covering the group widget tags).

Some common uses of group widgets in the UI system include:

- Skins, used for graphically skinning UI lists
- Assembling together the primary top level UI objects for a UI screen
- Assembling groups of display widgets together to be swapped in & out as the user scrolls through a UI list widget

List Item Widgets

List item widgets are used to provide a selection window into a list of items. They are analogous to the items in a drop-down menu in a Windows GUI for example. A list item may also invoke a submenu (this occurs if the datasource element referenced by a list item exports a valid value for the **submenu control name** key). These items are contained in a UI screen datasource object (more on this later) which is the core object used to tie the gameshell UI to game data. Like group widgets, list item widgets can respond to inputs and perform simple logic. The visual appearance of a list item is dictated by the list skin used by the list item's parent list widget. In addition to the common widget properties, a list item widget has the following custom properties:

- item label (string_id)— Child text widgets can reference this value to set their string contents using this value
- Event handlers

List Widgets

List widgets are the primary control widget in the gameshell UI. A list widget contains one or more list item widgets to provide a visible window into one of the UI screen's datasources. In addition to the common widget properties, a list widget has the following custom properties:

- **datasource name**— This is the name of the datasource contained in the parent screen widget to which this list refers
- **Tag reference to a skin collection**. This is a reference to the skin tag which will be applied to the list's item widgets for display pruposes
- One or more list item widgets
- Optional bitmap indicators which are used when there are more available items in the list than can be displayed at once in the list widget. They're used to indicate the existence of additional data items ahead and/or behind the currently displayed selection

Screen Widgets

The screen is the top level widget in the hierarchy. They will need to contain one or more group widgets in order to do anything interesting. It's through the screen widget that the rest of the game interacts with the gameshell UI (primarily through the UI message passing system and the UI window manager). In addition to the common widget properties, a screen widget has the following custom properties:

- Tag reference to a string list tag. This is used to contain any text strings that this UI screen might need to display specific to this UI screen (meaning that global or dynamic text strings could come from another location potentially)
- One or more datasource objects
- One or more button keys
- Tag references to sound tags which can be used to override the default gameshell UI sound effects
- Event handlers

Datasources

Datasources are the means by which a collection of interesting game data is stored and made available for display and interaction with the gameshell UI.

Conceptually a datasource can be thought of as a one-dimensional list of objects, each of which can have multiple interesting properties which it might want to make known to the UI.

Examples include:

- A list of players in the game
- The available control settings for a player profile
- All of the slayer game variants stored on a memory card
- The list of actions that a user can take at the main menu screen

Each element in a datasource can export named properties of itself in the following formats:

- A signed integer number
- A string_id
- Raw Unicode text string
- A string_id value used to indicate a submenu (in this case, the string_id name refers to the name assigned to a list widget in the active group)

An element may also export a string_id control name, meaning that if this element in the datasource is selected, the named UI control (submenu) should be invoked. The names associated with a datasource element property are string_ids. By hooking up other display widgets' properties to be driven by exported properties of a datasource, widgets can display a wide variety of dynamic content with very little effort required.

Animation

While gameshell UI is active and the user is interacting with it, various elements can change state (focused/unfocused, transitioning, finished transitioning). Widgets can also be in multiple states at the same time (e.g. transitioning into the screen while changing focus). The UI widget animation system has been designed to facilitate authoring animation tracks to be applied to widgets as they change states. The possible states that a widget can acquire include:

- **focused-ambient**— Widget is not transitioning between states and has input focus
- **unfocused-ambient**— Widget is not transitioning between states and does not have input focus
- **transition-from-screen**— This UI is transitioning away from the screen, going forward
- **transition-to-screen**— This UI is transitioning into the screen, going forward
- **transition-back-from-screen**— This UI is transitioning away from the screen, going backward
- **transition-back-to-screen**— This UI is transitioning into the screen, going backward
- **cycle-in-previous-display-pane**— This UI is on a pane that is previous to the currently active one, and is now being cycled in
- **cycle-in-next-display-pane**—This UI is on a pane that follows the currently active one, and is now being cycled in

- **cycle-out-previous-display-pane**—This UI is on the currently active pane, and is being cycled out to load the previous pane
- **cycle-out-next-display-pane**— This UI is on the currently active pane, and is being cycled out to load the next pane
- **list-display-group-transition-in**—This UI is in a group loaded as a list display group, and is being transitioned in
- **list-display-group-transition-out**— This UI is in a group loaded as a list display group, and is being transitioned out
- **control-received-focus**— This UI is part of a control which just received input focus
- **control-lost-focus**— This UI is part of a control which just lost input focus
- **indicator-ambient-additional-item**—This is used for a list's additional-items-indicator-bitmap widget, when not cycling through the list and there are additional items in its direction
- **indicator-ambient-no-additional-items**— This is used for a list's additional-items-indicator-bitmap widget, when not cycling through the list and there are no more additional items in its direction
- **indicator-activated-additional-items**— This is used for a list's additional-items-indicator-bitmap widget, when cycling through the list and there are more items in its direction
- **indicator-activated-no-additional-items**— This is used for a list's additional-items-indicator-bitmap widget, when cycling through the list and there are no more items in its direction
- **load-submenu**— This widget is part of a list which has just invoked a submenu
- **unload-submenu**— This widget is part of a list which has just unloaded a submenu
- **load-as-submenu**— This widget is part of a list that is being loaded as a submenu
- **unload-as-submenu**— This widget is part of a list loaded as a submenu which is now being unloaded
- **child-submenu-active-ambient**— This widget is part of a list who has a submenu active which has finished transitioning into view

For each of the above widget animation states, the following widget properies can be animated:

- Color
- Position
- Rotation about a point local to the widget
- Scale about a point local to the widget
- Texture coordinates (applicable to bitmaps only)
- Sprite sequence and frame (applicable to bitmaps only)
- Font (applicable to text only)

The animation of any particular property is performed by interpolating between a series of keyframes for that property, keyframes being defined by a property value and a time offset (from the time that the animation started).

The method of interpolation between keyframes is determined from the component animation. The animation has as "default" interpolation function which is applied to all keyframe interpolations, unless a keyframe has a custom interpolation function defined for it— in which case the animation will use that interpolation function when interpolating from that keyframe to another.

When defined in the animation tags, the initial (default) function for interpolations is an un-scaled (0-1) linear interpolation from the first keyframe value to the second keyframe value. Figure 1 depicts the color component animation tag for an animation using the default linear function, and no custom transition functions for the keyframes. By using the provided function types and curves, it's possible to get very complex and interesting animations with only a single pair of keyframes representing the starting and ending component values.

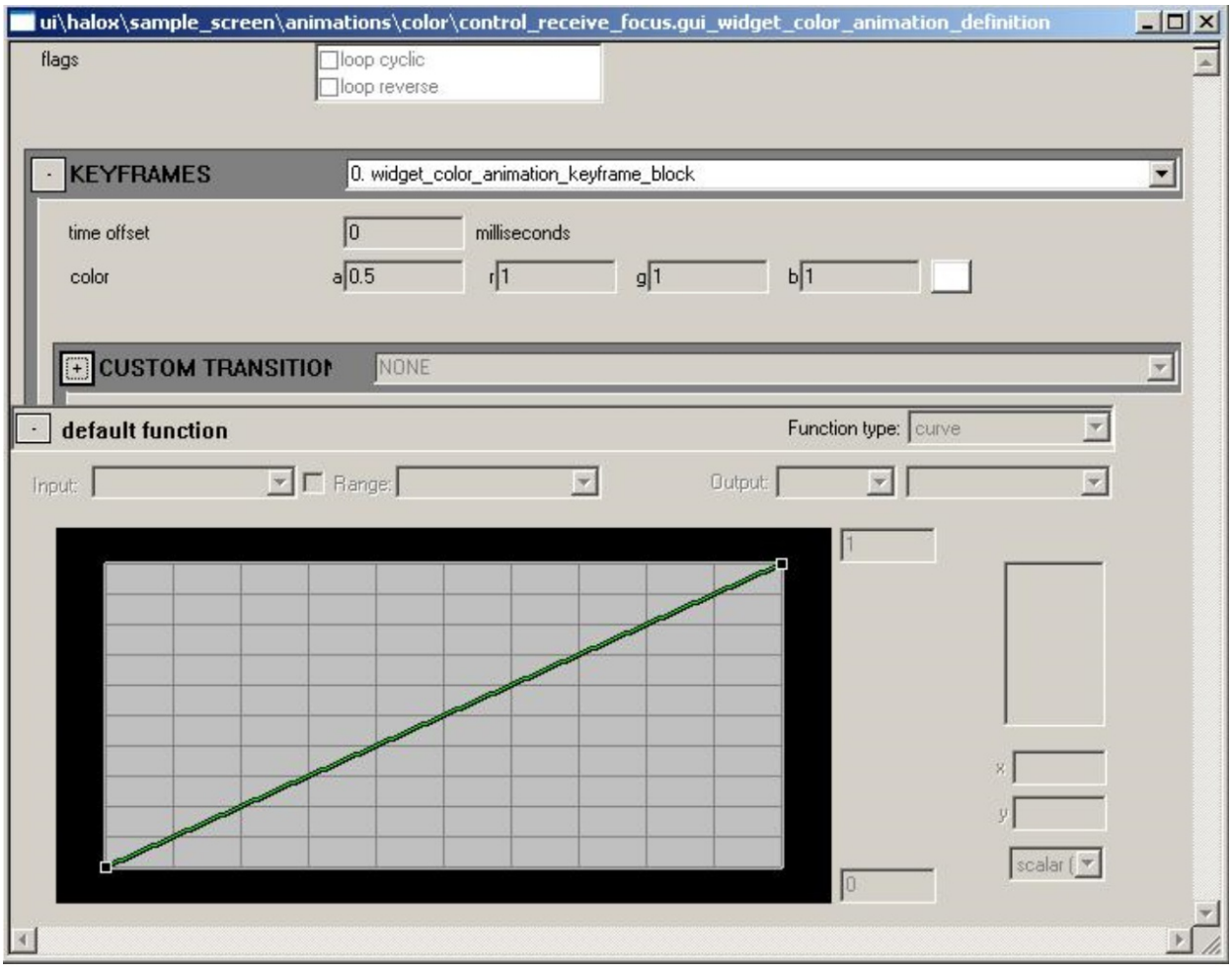


Figure 1 - Animation definition

The property values for a widget authored in tags is considered the baseline state. Animation of the widget applies the provided animations for any applicable states to this widget baseline state to provide the final animated, rendered state used when displaying the widget on screen.

Tag Structures

Presented here is a breakdown of the gameshell UI tag structures, beginning at the top with the UI globals tags and working down.

UI Globals Tag

There is a UI globals tag required for each of the three different scenario (map) types: main menu, single player, and multiplayer. At the bottom of this tag is a tag block where you add tag references to the UI screens that are to be made available for the respective scenario type. In order for a UI screen to properly load in a given scenario, a reference to its screen widget tag must be present in the corresponding UI globals tag. The **Halo 3** screen widget tags section is used for Halo3 UI screen widget tag references.

GUI Screen Widget Definition Tag

The screen widget definition tag is the top level tag for defining a UI screen. Here is a description of each of the fields in this tag:

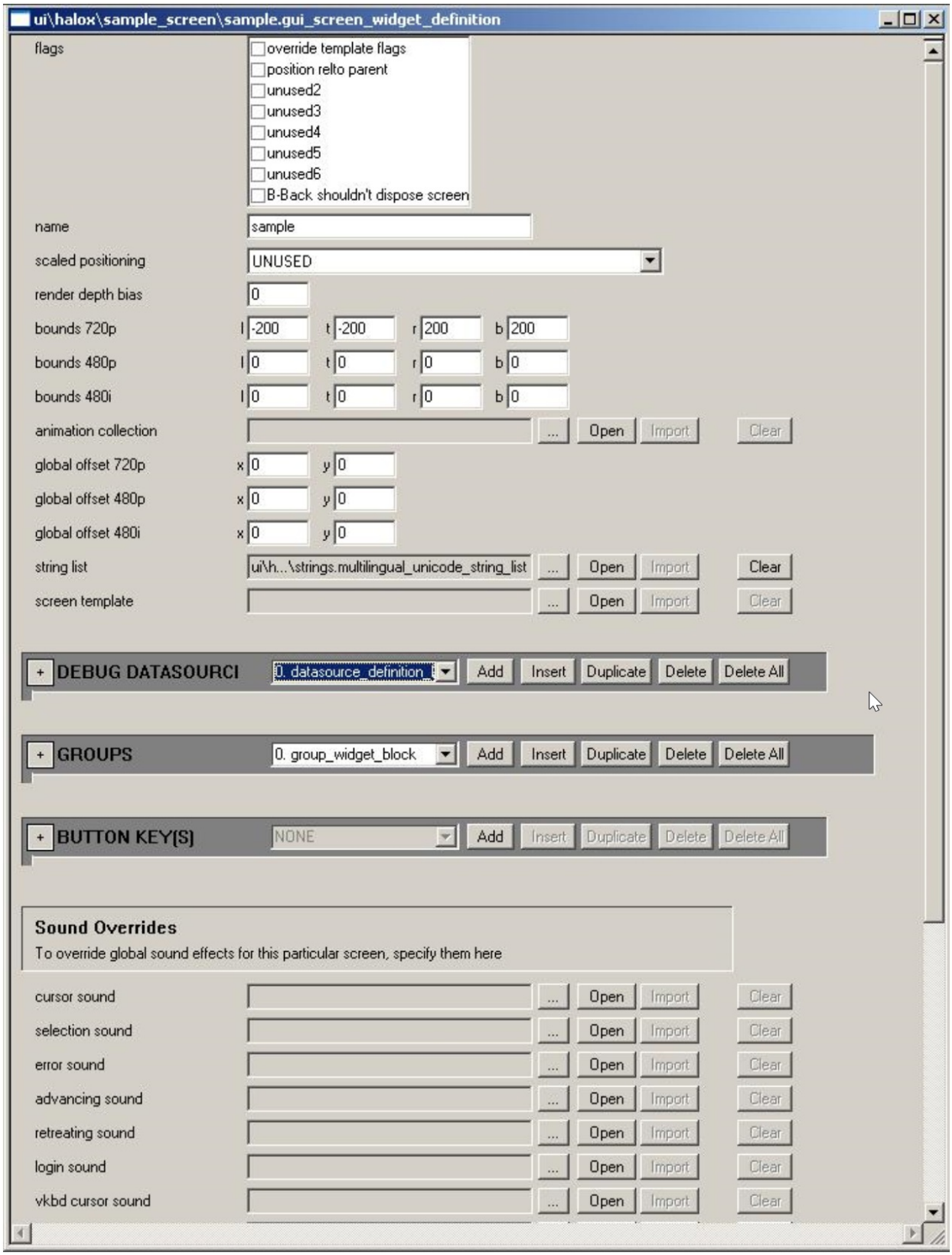


Figure 2 - GUI Screen Widget Defintion

- **flags**— Various behavioral properties
 - **override template flags**— If set, the flags here will override all template flags. Otherwise, they are combined with any template flags
 - **position relative to parent**— Ignored for screens
- **blback shouldn't dispose screen**— If set, automatic backing out of a screen via the B or Back buttons is disabled
- **name** — String_id name for this screen — must be set to some non-empty value in order for the screen tag to be usable

- **scaled positioning**— Defines how screen is automatically positioned for non-default display resolutions. This works as follows:
- If there is no authored bounds for the widget in the current display resolution, the bounds defined for the default authoring resolution (720 widescreen) are used.
- Next, the height and width are scaled horizontally and vertically by the ratio of the current display resolution to the default authoring resolution, respectively.
- Finally, based on the scaled positioning setting, the following happens:
 - **centered (default)**— Nothing else is done
 - **top edge**— The scaled bounds are aligned to the top edge of the default-authored bounds
 - **bottom edge**— The scaled bounds are aligned to the bottom edge of the default-authored bounds
 - **left edge**— The scaled bounds are aligned to the left edge of the default-authored bounds
 - **right edge**— The scaled bounds are aligned to the right edge of the default-authored bounds
 - **top-left**— The scaled bounds are aligned to the top and left edges of the default-authored bounds
 - **top-right**— The scaled bounds are aligned to the top and right edges of the default-authored bounds
 - **bottom-right**— The scaled bounds are aligned to the bottom and right edges of the default-authored bounds
 - **bottom-left**— The scaled bounds are aligned to the bottom and left edges of the default-authored bounds
- **render depth bias** — Ignored for screens
- **bounds 720p** — Absolute bounds for widget at 720p resolution
- **bounds 480p** — Custom bounds for widget at 480p resolution. Unused if 0
- **bounds 480i** — Custom bounds for widget at 480i resolution. Unused if 0
- **animation collection** — Tag reference to animations used for screen. Although these won't directly affect rendering, they can impact timing of transitions at the screen level, so it will be useful to use a set of dummy animations here in order to set time durations for transitions
- **global offset 720p** — A 2D global offset applied to all screen elements at 720p resolution
- **global offset 480p** — A 2D global offset applied to all screen elements at 480p resolution
- **global offset 480i** — A 2D global offset applied to all screen elements at 480i resolution
- **string list** — Reference to the multilingual unicode string list tag to be used for this screen's text
- **screen template** — Optional reference to a gui_screen_widget_definition tag to be used as a template for the screen
- **debug datasources** — Tag block of references to datasource tags, used for populating screen lists when testing a UI screen
- **groups** — Tag block of group widget definitions (see below for more details)
- **button keys** — Tag block of references to button key definition tags used for defining button key groups for use by the screen
- **sound overrides** — Optional tag references to sound tags which can be used to override the default UI sound effects

Datasource Tag

The datasource tag is used to populate a screen's lists when running in the debugging/test mode. They provide a means for the designer to populate a UI screen with virtual game data, so as to enable testing without any requirement for special code to provide actual game data to the screen elements. Following is a description of each of the fields in this tag:

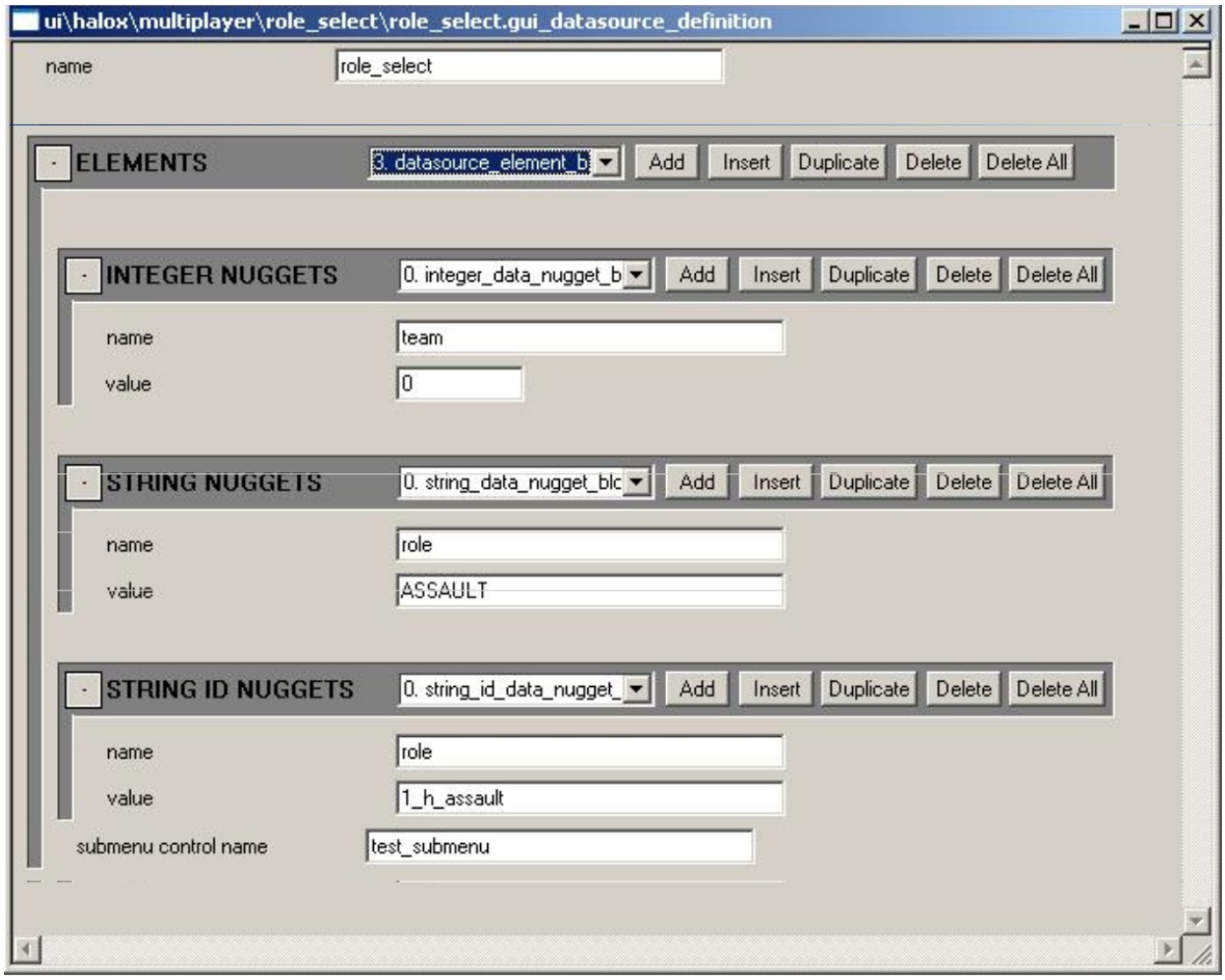


Figure 3 - Datasource Tag

- **name** — The string_id name for the datasource — must be an non-empty string in order for the datasource to be used
- **elements** — A tag block of entries for the datasource. Each element can have the following properties:
 - **integers** — A tag block of signed integer values. Each entry has a string_id name and a signed integer value
 - **strings** — A tag block of ASCII string values. Each entry has a string_id name and an ASCII string value
 - **string_ids** — A tag block of string_id values. Each entry has a string_id name and a string_id value
 - **submenu control name** — A string_id name of a list widget defined in the screen's primary group block definition which should be invoked if the list item representing this datasource element is selected by the user

Group Widget Definition

Group widgets can be defined as an instance in a screen widget definition, as well as a separate tag for use as a group widget instance template. The data fields in the tag for both are identical with the exception that in the separate tag used for templating, there is no field for a template tag reference (there is no recursive templating of widget definitions). Below is a description of the fields in a group widget tag definition:

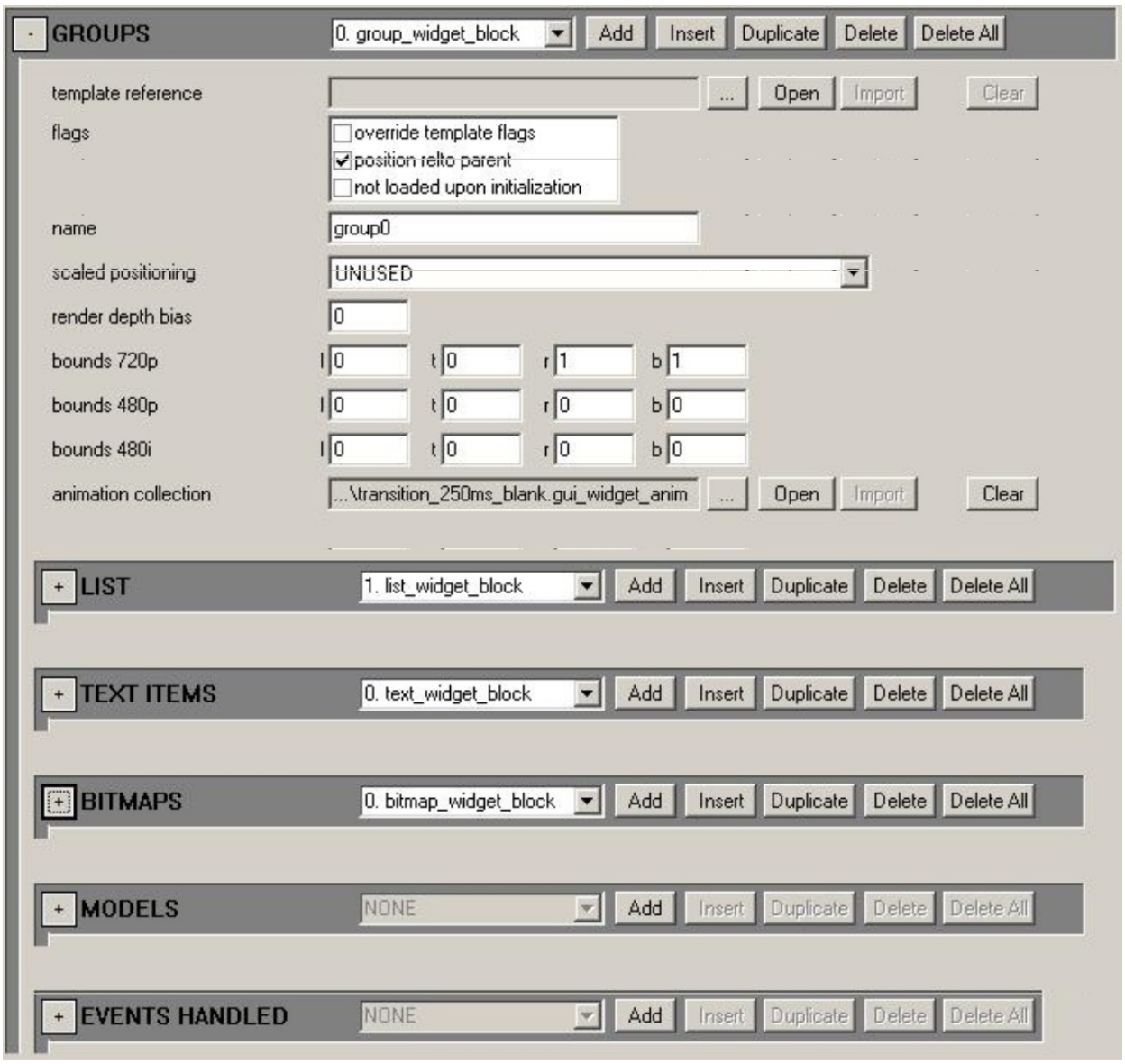


Figure 4 - Group Widget

- **template reference** — Found in the instance definition, this is an optional reference to a group widget definition tag to be used as a template for this group widget instance
- **flags**:
 - **override template flags** — If set, the flags here will override all template flags. Otherwise, they are combined with any template flags
 - **position relative to parent** — If set, the widget will position itself relative to the parent widget's position. Otherwise, it will be positioned in absolute window space
 - **don't load on initialization** — If this is set, the group and its children will not automatically load when the parent screen is loaded. Otherwise, the group and its children will load when the screen is loaded. The purpose of this flag is to allow the designer to author flavor groups for use when scrolling through a list menu
- **name** — This is the string_id name for the widget
- **scaled positioning** — Defines how widget is automatically positioned for non-default display resolutions
- **render depth bias** — Ignored for groups
- **bounds 720p** — Absolute bounds for widget at 720p resolution
- **bounds 480p** — Custom bounds for widget at 480p resolution. Unused if 0
- **bounds 480i** — Custom bounds for widget at 480i resolution. Unused if 0
- **animation collection** — Tag reference to animations used for the widget. Although these won't directly affect rendering, they can impact timing of transitions at the screen level, so it will be useful to use a set of

dummy animations here in order to set time durations for transitions

- **lists** — A tag block of list widget definitions. Any lists defined here and not flagged as being submenus are loaded when the group is loaded
- **text items** — A tag block of text widget definitions. Any text widgets defined here are loaded when the group is loaded
- **bitmaps** — A tag block of bitmap widget definitions. Any bitmap widgets defined here are loaded when the group is loaded
- **models** — A tag block of model widget definitions. Any model widgets defined here are loaded when the group is loaded
- **event handlers** — A tag block of event handlers. Any handlers defined here will be used by the group widget as appropriate

Text Widget Definition

Text widgets can be defined as an instance in a group, button key, or list skin, as well as a separate tag for use as a text widget instance template. The data fields in the tag for all cases are identical with the exception that in the separate tag used for templating, there is no field for a template tag reference (there is no recursive templating of widget definitions). Below is a description of the fields in a text widget definition:

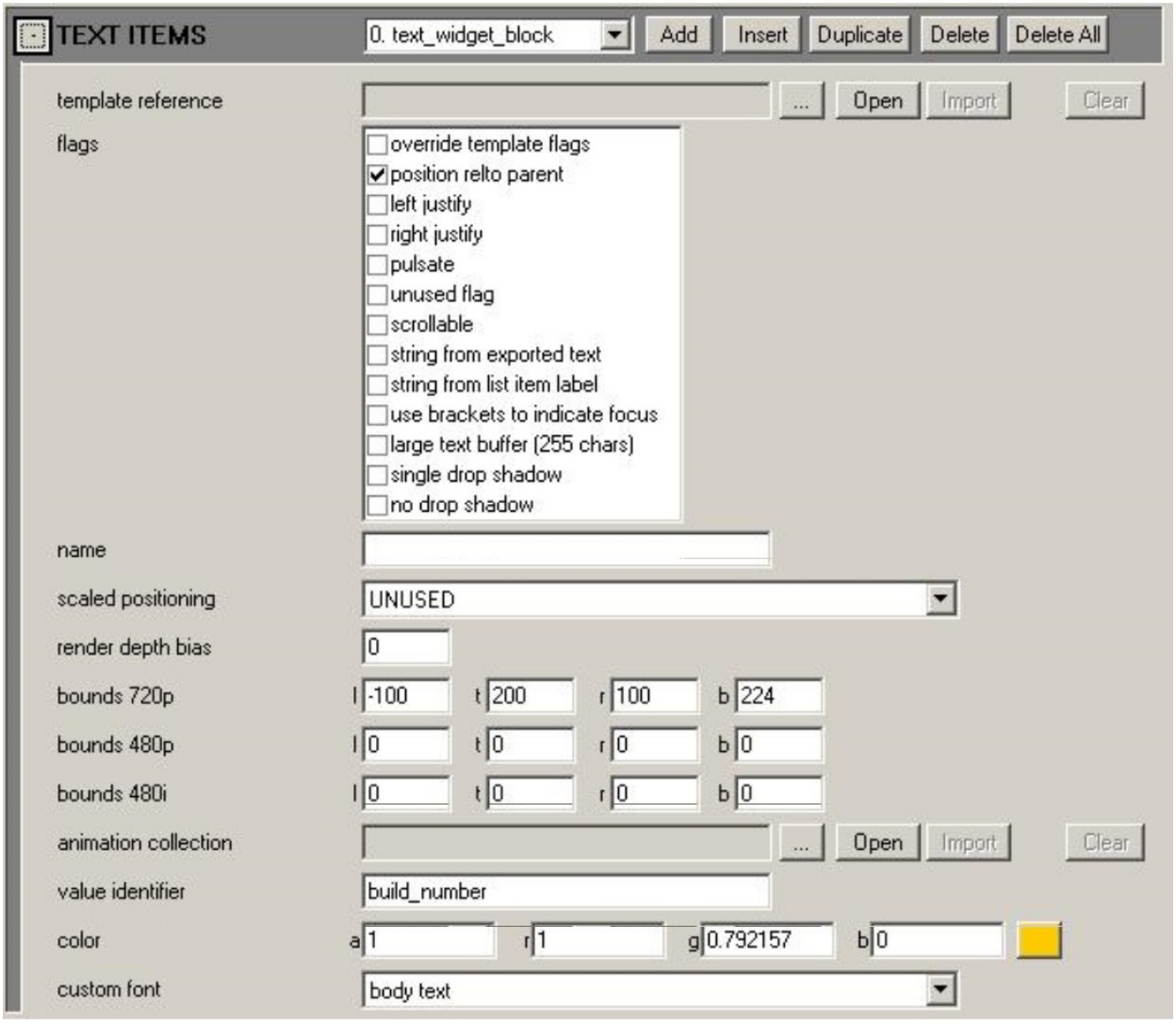


Figure 5 - Text Widget

- **template reference** — Found in any of the instance definitions, this is an optional reference to a text widget definition tag to be used as a template for this text widget instance
- **flags
 - **override template flags** — If set, the flags here will override all template flags. Otherwise, they are

combined with any template flags
 - **position relative to parent** — If set, the widget will position itself relative to the parent widget's position. Otherwise, it will be positioned in absolute window space
 - **left justify** — If set, text is left justified (default is centered)
 - **right justify** — If set, text is right justified (default is centered)
 - **pulsate** — If set, text will automatically pulsate
 - **scrollable** — Currently not implemented
 - **string from exported text** — If set, the text widget will attempt to set its contents according to the string value exported from its parent list item for the value name specified in the text widget's "value identifier" field
 - **string from list item label** — If set, the text widget will attempt to set its contents according to the string id value for its parent list item "item label" field
 - **use brackets to indicate focus** — If set, when the widget has focus, its text contents will be wrapped by a pair of square bracket characters
 - **large text buffer** — Set this flag if you want the widget to accommodate strings up to 255 characters (the default limit is 31 characters)
 - **single drop shadow** — Set this flag if you want the text to render with a single drop shadow offset down and to the right one texel (default is to be outlined by 1 texel)
 - **no drop shadow** — Set this flag if you want the text to render with no drop shadow at all
- **name** — This is the string_id name for the widget
- **scaled positioning** — Defines how widget is automatically positioned for non-default display resolutions
- **render depth bias** — A biasing value used when rendering, if there are multiple widgets at the same exact three-dimensional depth
- **bounds 720p** — Absolute bounds for widget at 720p resolution
- **bounds 480p** — Custom bounds for widget at 480p resolution— unused if 0
- **bounds 480i** — Custom bounds for widget at 480i resolution— unused if 0
- **animation collection** — Tag reference to animations used for the widget
- **value identifier** — This is the string_id key used when setting the text string from either an exported datasource value, or from the screen's string list tag (the default behavior). This will always need to be set in order for any text to display
- **color** — The baseline color for the text glyphs
- **custom font** — The baseline font for the text glyphs

Bitmap Widget Definition

Bitmap widgets can be defined as an instance in a group, button key or list skin, as well as a separate tag for use as a bitmap widget instance template. The data fields in the tag for all cases are identical with the exception that in the separate tag used for templating, there is no field for a template tag reference (there is no recursive templating of widget definitions). Below is a description of the fields in a bitmap widget definition:

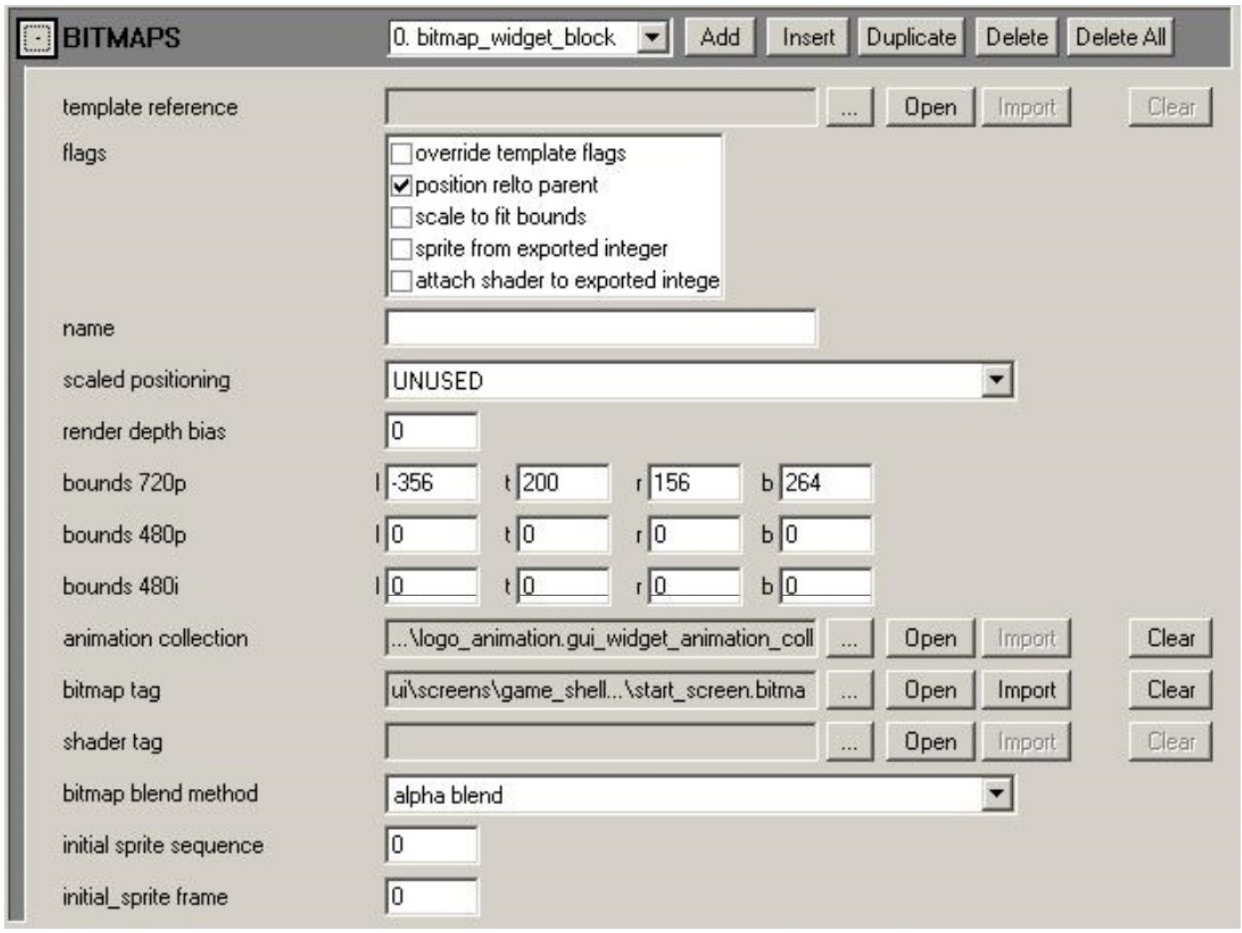


Figure 6 - Bitmap Widget

- **template reference** — Found in any of the instance definitions, this is an optional reference to a bitmap widget definition tag to be used as a template for this bitmap widget instance
- **flags**:
 - **override template flags** — If set, the flags here will override all template flags. Otherwise, they are combined with any template flags
 - **position relative to parent** — If set, the widget will position itself relative to the parent widget's position. Otherwise, it will be positioned in absolute window space
 - **scale to fit bounds** — If set, the bitmap will be stretched to conform to the animated, defined bounds. Otherwise, it will prefer it's actual pixel bounds
 - **sprite from exported integer** — If set, the bitmap will try to set its sprite frame index to the integer value exported from a parent list item for the value with the key name with the same string_id value as this bitmap widget's "name" field
 - **attach shader to exported integer** — Unused
- **name** — This is the string_id name for the bitmap, also used for retrieving exported values if necessary
- **scaled positioning** — Defines how widget is automatically positioned for non-default display resolutions
- **render depth bias** — A biasing value used when rendering, if there are multiple widgets at the same exact three-dimensional depth
- **bounds 720p** — Absolute bounds for widget at 720p resolution
- **bounds 480p** — Custom bounds for widget at 480p resolution— unused if 0
- **bounds 480i** — Custom bounds for widget at 480i resolution— unused if 0
- **animation collection** — Tag reference to animations used for the widget
- **bitmap tag** — Tag reference to a bitmap widget containing the sprites to be used when rendering
- **shader tag** — Presently unused
- **bitmap blend method** — The render blending method used when rendering sprites

- **initial sprite sequence** — The baseline sprite sequence to use when rendering sprites
- **initial sprite frame** — The baseline sprite frame to use when rendering sprites

List Widget Definition

List widgets can be defined as an instance in a group as well as a separate tag for use as a list widget instance template. The data fields in the tag for both cases are identical— with the exception that in the separate tag used for templating, there is no field for a template tag reference (there is no recursive templating of widget definitions). Below is a description of the fields in a list widget definition:

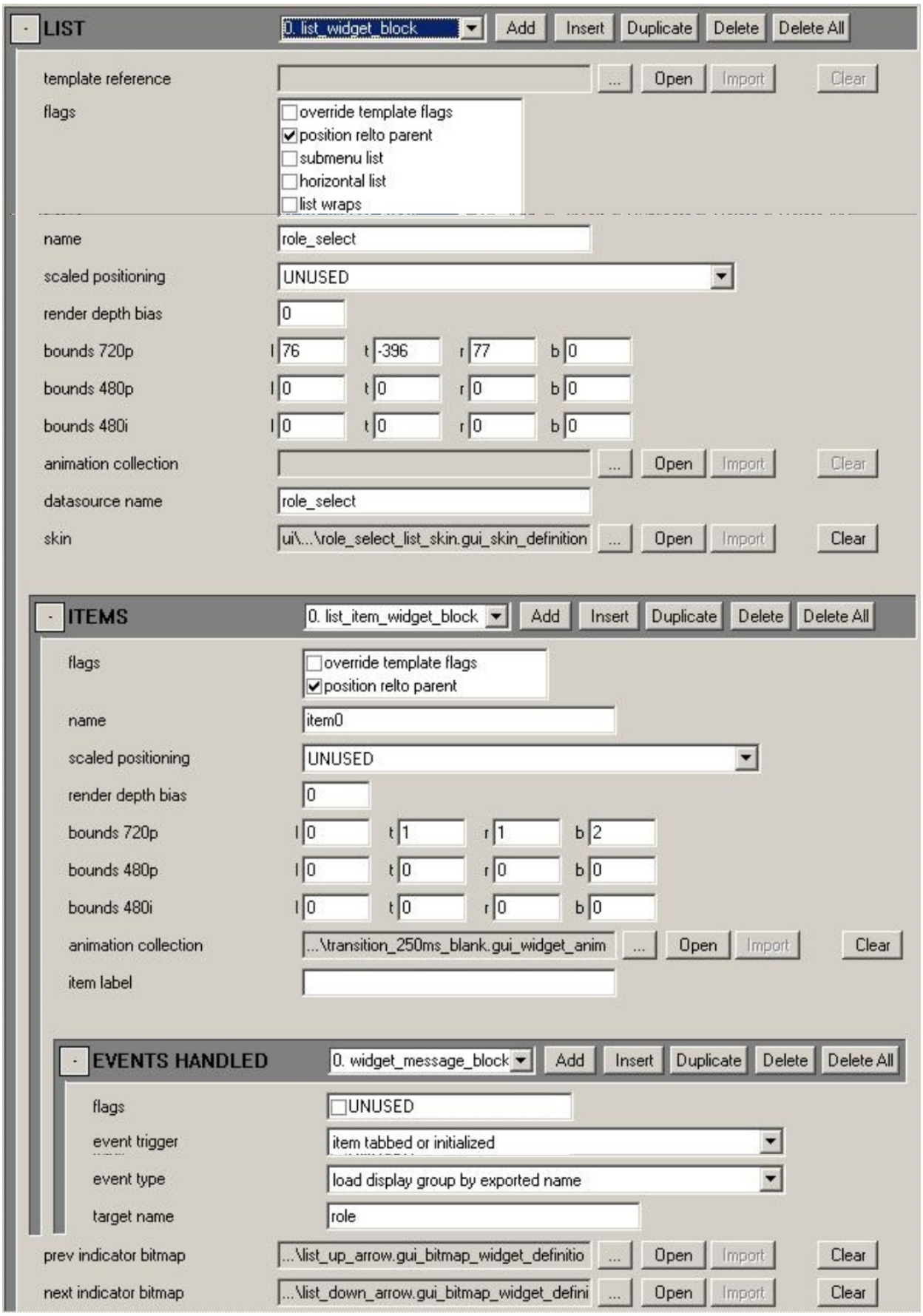


Figure 7 - List Widget

- **template reference** — Found in the instance definitions, this is an optional reference to a list widget definition tag to be used as a template for this list widget instance
- **flags**:
 - **override template flags** — If set, the flags here will override all template flags. Otherwise, they are combined with any template flags

 - **position relative to parent** — If set, the widget will position itself relative to the parent widget's position. Otherwise, it will be positioned in absolute window space
 - **scale to fit bounds** — If set, the bitmap will be stretched to conform to the animated, defined bounds. Otherwise, it will prefer it's actual pixel bounds
 - **horizontal list** — If set, tabbing left/right cycles through the list items. Otherwise, tabbing up/down cycles through list items
 - **list wraps** — If set, the list wraps around when scrolling. Otherwise, the list does not wrap
- **name** — This is the string_id name for the list
- **scaled positioning** — Defines how widget is automatically positioned for non-default display resolutions
- **render depth bias** — Ignored for lists
- **bounds 720p** — Absolute bounds for widget at 720p resolution
- **bounds 480p** — Custom bounds for widget at 480p resolution— unused if 0
- **bounds 480i** — Custom bounds for widget at 480i resolution — unused if 0
- **animation collection** — Tag reference to animations used for the widget. Although these won't directly affect rendering, they can impact timing of transitions, so it will be useful to use a set of dummy animations here in order to set time durations for transitions
- **datasource name** — String_id name of the desired screen datasource to use
- **skin** — Reference to a UI list skin tag, which is used to populate list items
- **items** — Tag block where the list's item widgets are defined. The parameters for list items are as follows:
 - **flags** —
 - override template flags: Unused for list item widgets (these cannot be templated)
 - position relative to parent: If set, the widget will position itself relative to the parent widget's position. Otherwise, it will be positioned in absolute window space
 - **name** — This is the string_id name for the list item widget
 - **scaled positioning** — Defines how widget is automatically positioned for non-default display resolutions
 - **render depth bias** — Ignored for list items
 - **bounds 720p** — Absolute bounds for widget at 720p resolution
 - **bounds 480p** — Custom bounds for widget at 480p resolution — unused if 0
 - **bounds 480i** — Custom bounds for widget at 480i resolution — unused if 0
 - **animation collection** — Tag reference to animations used for the widget. Although these won't directly affect rendering, they can impact timing of transitions, so it will be useful to use a set of dummy animations here in order to set time durations for transitions
 - **item label** — **String_id** value that can be used by child text widgets for the purpose of labelling the item
 - **event handlers** — A tag block of event handlers. Any handlers defined here will be used by the list item widget as appropriate
- **previous indicator bitmap** — Reference to a bitmap widget tag which is used for indicating additional menu items preceeding the visible selection
- **next indicator bitmap** — Reference to a bitmap widget tag which is used for indicating additional menu items following the visible selection

List Skin Definition

List skin definitions are used to specify how a list item will populate itself with rendered flavor widgets such as text and bitmaps. Below is a description of the fields in a list widget definition:

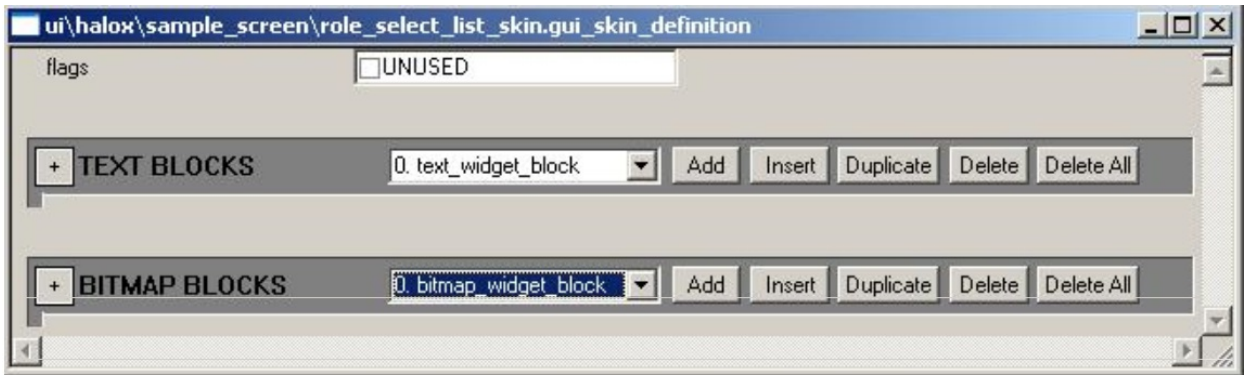


Figure 8 - List Skin

- **flags** — Currently unused
- **text blocks** — These are definitions for text widgets that will be added as part of the skin
- **bitmaps** — These are definitions for bitmap widgets that will be added as part of the skin

Component Animation Definition

Widget animation component definitions (see Figure 1) are used to define how to animate a single component of the widget's rendered appearance. There are component animation definitions for the following animatable propertes: color, position, scale, rotation, texture coordinates, sprite frame, and text font. Below is a description of the fields in a widget component animation definition:

- **flags** — Use to specify the looping style for the animation (all unchecked means play through once)
- **keyframes** — Tag block for the animation keyframes. There need to be at least two keyframes for the animation to be valid. Keyframes consist of a time offset in milliseconds (should be increasing each keyframe) and a property value for that keyframe. Additionally, there is a tag block where you can define a custom interpolation function to be used when interpolating from this keyframe value to the next keyframe value. If no custom interpolation function is defined for a keyframe, it uses the animation's default interpolation function
- **default function** — This is an editable function which allows you to define how to interpolate between keyframe values for keyframes that do not have a custom interpolation function. By default, this function will be a simple linear interpolation with no scaling

Widget Animation Definition

Widget animation definitions are used to collect all the desired component animations for a single widget state animation together into one tag. Below is a description of the fields in a widget animation definition:

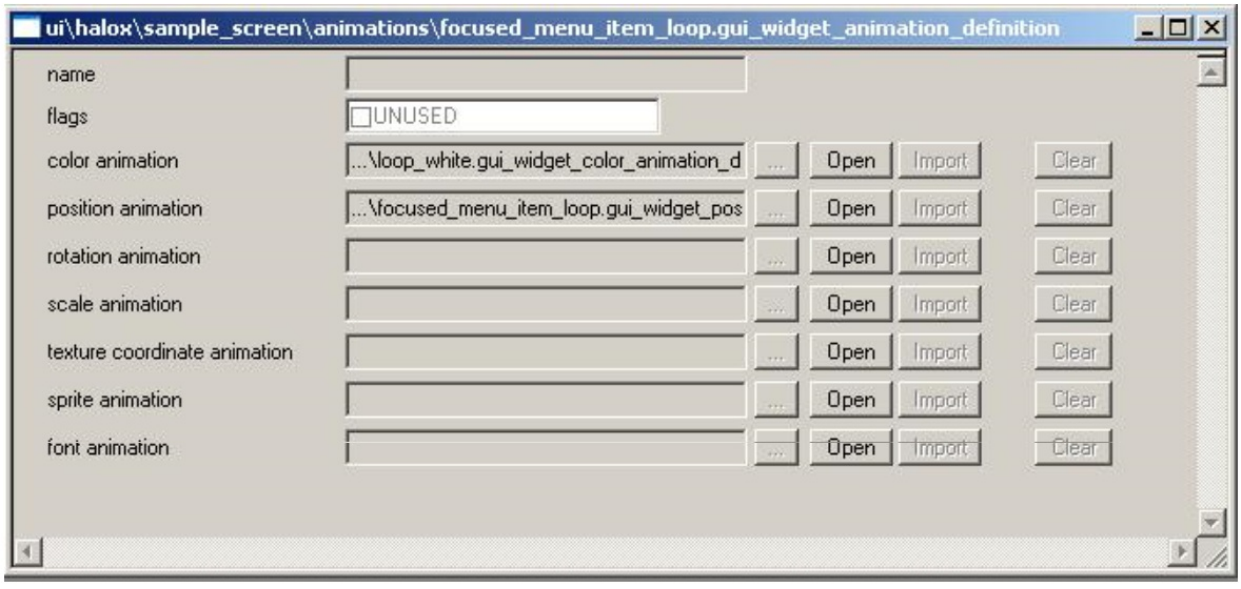


Figure 9 - Animation Definition

- **name** — Currently unused
- **flags** — Currently unused
- **color animation** — Optional tag reference to color component animation tag
- **position animation** — Optional tag reference to position component animation tag
- **rotation animation** — Optional tag reference to rotation component animation tag
- **scale animation** — Optional tag reference to scale component animation tag
- **texture coordinate animation** — Optional tag reference to texture coordinate component animation tag
- **sprite animation** — Optional tag reference to sprite component animation tag
- **font animation** — Optional tag reference to font component animation tag

Widget Animation Collection Definition

Widget animation collection definitions are used to collect together all of the desired widget animations for each animation state into a single tag. This is the animation tag referenced by widgets. A widget's animations are determined by the animations referenced in the widget's animation collection. Below is a description of the fields in a widget animation collection definition:

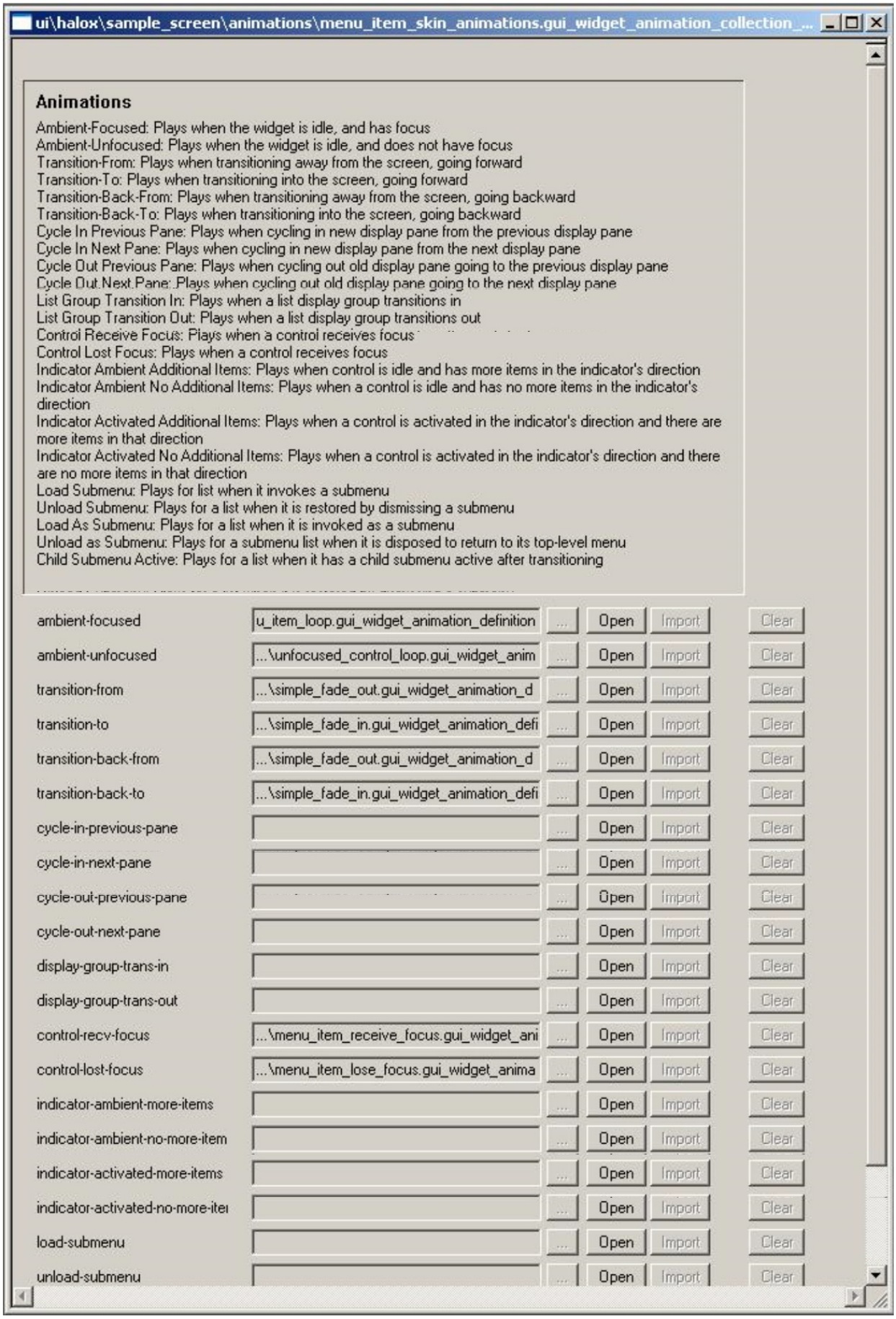


Figure 10 - Animation Collection

- **ambient-focused** — Optional animation to be applied when widget has input focus and is idle
- **ambient-unfocused** — Optional animation to be applied when widget does not have input focus and is idle
- **transition-from** — Optional animation to be applied when widget's screen is transitioning out, going forward

- **transition-to** — Optional animation to be applied when widget's screen is transitioning in, going forward
- **transition-back-from** — Optional animation to be applied when widget's screen is transitioning out, going backward
- **transition-back-to** — Optional animation to be applied when widget's screen is transitioning in, going backward
- **cycle-in-previous-pane** — Optional animation to be applied when widget's screen is cycling in as a preceding pane (screen)
- **cycle-in-next-pane** — Optional animation to be applied when widget's screen is cycling in as a following pane (screen)
- **cycle-out-previous-pane** — Optional animation to be applied when widget's screen is cycling out to a preceding pane (screen)
- **cycle-out-next-pane** — Optional animation to be applied when widget's screen is cycling out to a following pane (screen)
- **display-group-trans-in** — Optional animation to be applied when widget is part of a list display group that is transitioning in
- **display-group-trans-out** — Optional animation to be applied when widget is part of a list display group that is transitioning out
- **control-recv-focus** — Optional animation to be applied when widget is part of a control which has received input focus
- **control-lost-focus** — Optional animation to be applied when widget is part of a control which has lost input focus
- **indicator-ambient-more-items** — Optional animation to be applied when widget is an indicator bitmap for a list, the list is idle, and there are more items in the direction of the indicator
- **indicator-ambient-no-more-items** — Optional animation to be applied when widget is an indicator bitmap for a list, the list is idle, and there are no more items in the direction of the indicator
- **indicator-activated-more-items** — Optional animation to be applied when widget is an indicator bitmap for a list, the list is being scrolled in the indicator's direction, and there are more items in the direction of the indicator
- **indicator-activated-no-more-items** — Optional animation to be applied when widget is an indicator bitmap for a list, this list is being scrolled in the indicator's direction, and there are no more items in the direction of the indicator
- **load-submenu** — Optional animation to be applied when widget is part of a list, and the list is invoking a submenu control
- **unload-submenu** — Optional animation to be applied when widget is part of a list, and the list has a submenu control which is closing
- **load-as-submenu** — Optional animation to be applied when widget is part of a list which is being loaded as a submenu control
- **unload-as-submenu** — Optional animation to be applied when widget is pare of a list which was loaded as a submenu control, and is now being unloaded
- **child submenu active ambient** — Optional animation to be applied when widget is part of a list which has a submenu control loaded, and the submenu control is not transitioning in or out

How Templating Works

Templating of widgets is a powerful way to reuse common widget properties among several instances of like-typed widgets which, in their respective instances, differ in only a couple of ways from each other. All widget types support templating except for list items (that is the purpose of list skins, whose building blocks can themselves be templated). When a widget instance uses a template tag, the widget constructs itself in the following manner:

1. It reads the properties of the template tag.
2. Next it looks at each tag field property in the templated instance.
3. If the instanced property is anything different than the baseline (initial, zeroed) state, then the instanced property value is used in place of the template property value.

Figure 11 is an example of a text widget instance in a list skin which uses a text widget template. The template has been setup with some flags, a render depth bias, positioning, animations, color, and font. Notice that there has been nothing set here which will cause a text widget constructed with this definition to actually populate itself with any text string (it is not being set by any exported data, and its value identifier property has not been set).

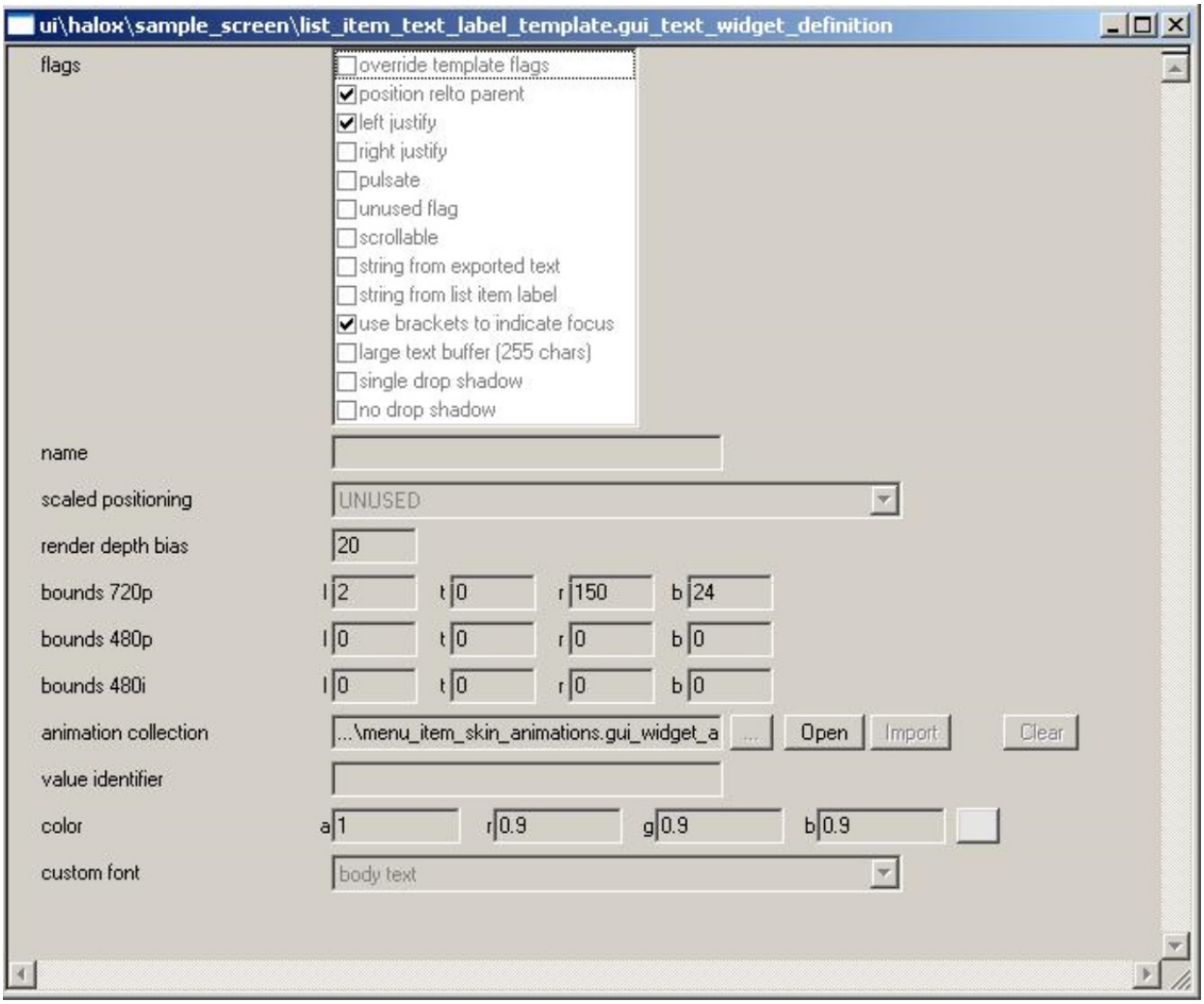


Figure 11 - Text Widget Instance Example

Figure 12 is an instance of a text widget in a list skin which is referencing the above text widget definition as a template (the template reference field has been set to the text widget definition described above). In the instance, only one flag has been set: **string from exported text**. As described earlier, this flag value will be combined with the flag values from the template. Therefore, this text widget instance is going to attempt to set its string contents according to data exported by its parent list item. The only other property which has been specified in this instance is the value identifier string_id, which has been set to role. Therefore, the datasource property that this text widget instance will attempt to extract has the property name role.

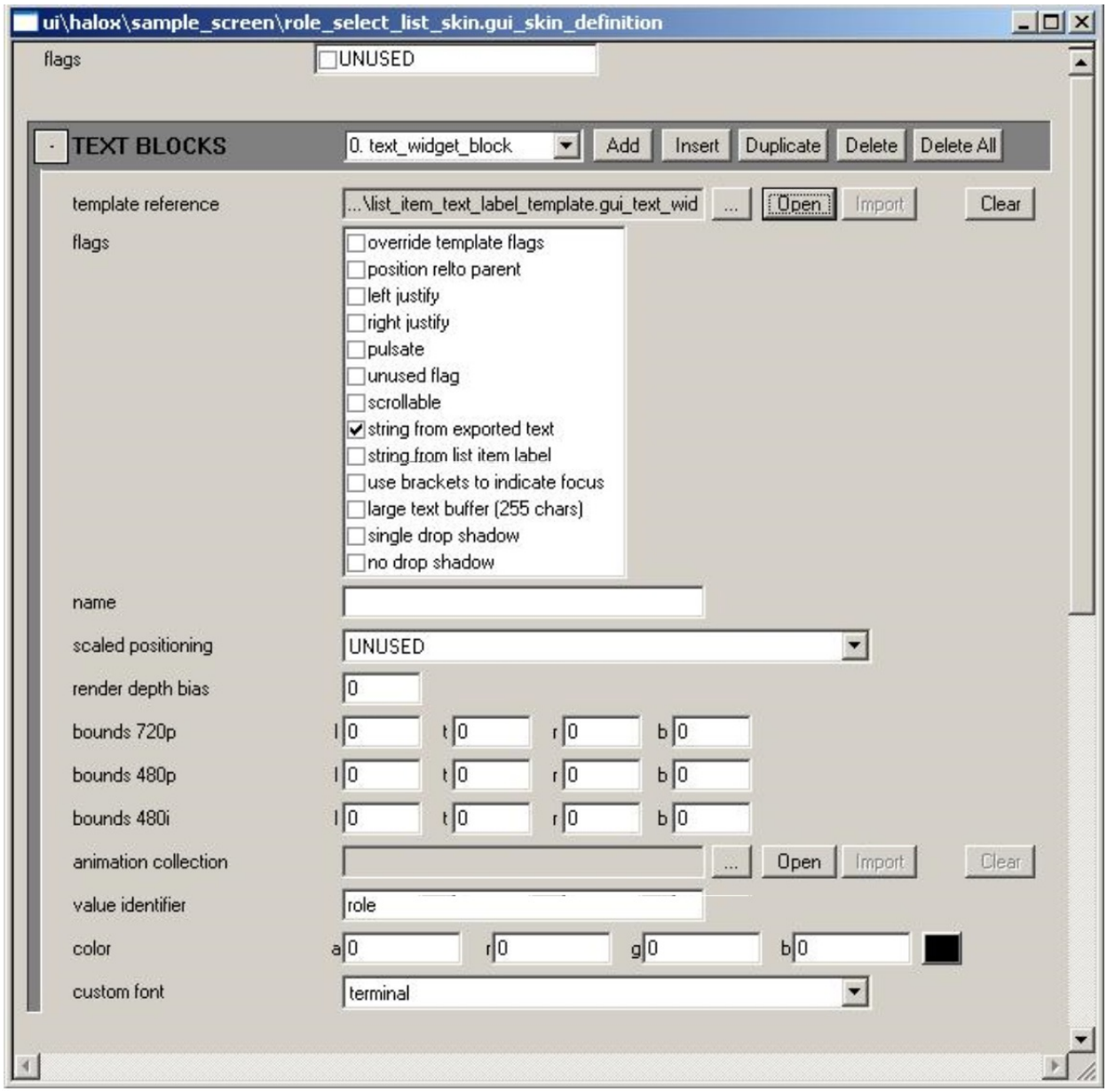


Figure 12 - Text Widget Instance Example

The benefit of using a template here is that several list skins could be quickly constructed to have a similar appearance, with the only difference in each instance being the source of the label text.

Debugging Commands and Variables

Below is a list of debugging/console commands pertaining to the gameshell UI and a description of their functionality.

- **ui_time_scale [numeric value]** — Use this to alter the speed with which time passes in the UI (e.g. 0.5 to run the UI at half-speed)
- **ui_display_memory [true|false]** — If set to true, the gameshell UI system's memory usage is displayed in real time on the screen
- **gui_debug_text_bounds_global [true|false]** — If set to true, debug outlines of all text widgets will be rendered
- **gui_debug_bitmap_bounds_global [true|false]** — If set to true, debug outlines of all bitmap widgets will be rendered
- **gui_debug_list_item_bounds_global [true|false]** — If set to true, debug outlines of all list item widgets will be rendered
- **gui_debug_list_bounds_global [true|false]** — If set to true, debug outlines of all list widgets will be

rendered

- **gui_debug_group_bounds_global [true|false]** — If set to true, debug outlines of all group widgets will be rendered
- **gui_debug_screen_bounds_global [true|false]** — If set to true, debug outlines of all screen widgets will be rendered
- **ui_debug_text_bounds [true|false]** — If set to true, debug outlines of all interface text will be rendered (more inclusive than gui_debug_text_bounds_global)
- **ui_debug_text_font [true|false]** — If set to true, all interface text renders the name of the font it is setup to use, rather than the normal string contents
- **ui_debug_show_title_safe_bounds [true|false]** — If set to true, the title safe boundaries are displayed in real time on the screen
- **ui_memory_dump [file name]** — Run this command to write the current UI memory useage information to the specified file (information includes details on all current allocations)
- **gui_test_screen [name]** — Run this command to load the specified UI screen in test mode
- **gui_load_screen_with_support_code [name]** — Run this command to load the specified UI screen using its custom support code
- **gui_print_active_screens** — Run this command to print a list of active UI screens to the display
- **gui_debug_screen_name [name] [on|off]** [recursive true|false] — Toggle display of given screen widget's name, optionally recursive
- **gui_debug_screen_animation [name] [on|off]** [recursive true|false] — Toggle display of given screen animations, optionally recursive
- **gui_debug_screen_bounds [name] [on|off]** [recursive true|false] — Toggle display of given screen's bounds, optionally recursive
- **gui_debug_screen_rotation [name] [on|off]** [recursive true|false] — Toggle display of given screen's rotation, optionally recursive
- **gui_debug_group_name [name] [on|off]** [recursive true|false] — Toggle display of given group's name, optionally recursive
- **gui_debug_group_animation [name] [on|off]** [recursive true|false] — Toggle display of given group's animations, optionally recursive
- **gui_debug_group_bounds [name] [on|off]** [recursive true|false] — Toggle display of given group's bounds, optionally recursive
- **gui_debug_group_rotation [name] [on|off]** [recursive true|false] — Toggle display of given group's rotation, optionally recursive
- **gui_debug_list_name [name] [on|off]** [recursive true|false] — Toggle display of given list's name, optionally recursive
- **gui_debug_list_animation [name] [on|off]** [recursive true|false] — Toggle display of given list's animations, optionally recursive
- **gui_debug_list_bounds [name] [on|off]** [recursive true|false] — Toggle display of given list's bounds, optionally recursive
- **gui_debug_list_rotation [name] [on|off]** [recursive true|false] — Toggle display of given list's rotation, optionally recursive
- **gui_debug_list_item_name [name] [on|off]** [recursive true|false] — Toggle display of given list item's name, optionally recursive
- **gui_debug_list_item_animation [name] [on|off]** [recursive true|false] — Toggle display of given list item's animation, optionally recursive
- **gui_debug_list_item_bounds [name] [on|off]** [recursive true|false] — Toggle display of given list item's bounds, optionally recursive
- **gui_debug_list_item_rotation [name] [on|off]** [recursive true|false] — Toggle display of given list item's rotation, optionally recursive

- **gui_debug_text_name [name] [on|off]** — Toggle display of given text widget's name
- **gui_debug_text_animation [name] [on|off]** — Toggle display of given text widget's animation state
- **gui_debug_text_bounds [name] [on|off]** — Toggle display of given text widget's bounds
- **gui_debug_text_rotation [name] [on|off]** — Toggle display of given text widget's rotation
- **gui_debug_bitmap_name [name] [on|off]** — Toggle display of given bitmap widget's name
- **gui_debug_bitmap_animation [name] [on|off]** — Toggle display of given bitmap widget's animation state
- **gui_debug_bitmap_bounds [name] [on|off]** — Toggle display of given bitmap widget's bounds
- **gui_debug_bitmap_rotation [name] [on|off]** — Toggle display of given bitmap widget's rotation

Miscellaneous

The following are some miscellaneous user interface related items that might be useful to know.

The UI Coordinate System

The coordinate system for the UI has its origin at the center of the render window (which is the entire display for full-screen windows, half of the display for 2-way split screen, and a quarter of the display for 4-way split screen). X is positive to the right, Y is positive down, and Z is positive going into the screen.

Inline Text Formatting

Text displayed in the UI is run through an XML parser prior to rendering. One of the things you can do is to apply inline formatting and color changes to text through the use of HTML-like tags. Below is a list of the XML text formatting tags currently supported:

- **HTML comment, format** — <!-- comment text --> Anything inside of an HTML comment tag is ignored for rendering purposes.
- **line break, format** —
 The line break tag causes a line break to be inserted at this point in the text string.
- **paragraph tag, format** — <p [align=left|center|right] /> The paragraph tag causes a double line break to be inserted at this point in the text string. If the align sub-tag is used, it causes the text justification to be changed as specified (left = left-justified, center = center-justified, right = right-justified).
- **paragraph close tag, format** — </p> The paragraph close tag causes a double line break to be inserted at this point in the text string, and also reverts any justification changes caused by a previous formatting tag.
- **division tag, format** — <div [align=left|center|right]> The division tag causes a line break to be inserted at this point in the text string. If the align sub-tag is used, it causes the text justification to be changed as specified (left = left-justified, center = center-justified, right = right-justified).
- **division close tag, format** — The division close tag causes a line break to be inserted at this point in the text string and also reverts any justification changes caused by a previous formatting tag.
- **font tag, format** — The font tag causes the text font to be changed to the specified font at this point in the text string. The valid font names are: terminal, body_text, title, super_large, large_body, split_screen_hud_message, full_screen_hud_message, english_body_text, hud_number, subtitle, and main_menu.
- **font close tag, format** — The font close tag causes the text font to be restored to the setting prior to the last tag encountered, at this point in the text string.
- **color tag, format** — <color argb=#AARRGGBB> The color tag causes the text color to be changed to the specified 32-bit hexidecimal ARGB value at this point in the text string.
- **color close tag, format** — </color> The color close tag causes the text color to be restored to the setting prior to the last <color> tag encountered, at this point in the text string.

Binding UI Screens to Classes in Source Code

The following describes how screen tags are bound to custom gui screen classes in the source code.

- First, define the custom screen class which will be used to drive this UI screen. All custom screen classes must derive from c_gui_screen_widget.
- Once the screen class has been defined, it can be bound to drive a specific screen tag by adding an entry to the screen binding list found in the header file c_gui_screen_widget_registration.h. This header file is self-explanatory, but for completeness' sake the steps to adding a new binding are:
 1. Include the header file for the custom screen class
 2. Add a registration entry for the custom screen class to a string_id (the string_id being the screen tag's name)
- With these changes, the game will now always use the corresponding custom screen class to drive the tags for the correspondingly named screen tag whenever it is loaded by the game (with the exception of loading screens in test-mode, in which case the generic test screen class is used to drive the UI screen).

HaloScript Overview

11/23/2022 • 20 minutes to read

HaloScript is the scripting language used within the game engine to make everything happen throughout the course of a mission. To generate a complete list of HS commands, open Sapien and type script_doc from the command console in the game window. A text document listing every command will be generated and placed in your \main folder.

Script Types

Startup Scripts

This is the starting point of every mission in the script file. There is only one startup script per mission. This script will execute all of its commands sequentially and is normally used to wake other dormant scripts and get things rolling for the mission. A startup script is named like the example below:

(script startup example_script)

Dormant Scripts

The most commonly used type of script. As the name implies, they are "dormant" until another script "wakes" them using the wake command. This is an instant command— the script that uses this command will not be blocked (or delayed).

Example

```
(wake example_script)
```

A dormant script can be seen as a separate thread when it is started. It will sequentially go through all of its commands until it is finished. There can only be one instance of that script at any time: it is not possible to wake the same dormant script two times in a row. Once a dormant script is finished, it is impossible to wake it again. If you want to wake a dormant script multiple times, consider using a static script instead, or use a continuous script with the following structure:

```
(script continuous looping_script
(sleep_forever)
(print "Do your things here")
)
```

You can stop a dormant script by using the **sleep_forever** command.

Static Scripts

This form of script is used to develop functions with a specific goal. They can be used by different dormant scripts or reused by the same dormant script. They are blocking, which means that a thread calling a static script that takes 5 seconds to execute will have to wait the entire 5 seconds before resuming its other activities. Basically, calling a static script is the equivalent of copy/pasting the entire static script where the call has been made.

Static scripts can return values and accept arguments. If you want a function to tell you if three objects are destroyed, you will probably want it to return a boolean, as shown here:

```
(script static boolean AreThreeObjectsDown
    ; Evaluate if the three objects are down
    (if (;* condition *;)
    true
    false
    )
)
```

The above script returns either true or false depending on the condition tested.

If you want a static script to have different behaviors depending on where you call it in your script, you can pass arguments to it. If, for example, you want a static script to send debug information to the screen when you specify it, you can use the following static script structure:

```
(script static void (do_something (boolean debug_print))
    ;Do something
    (if (= debug_print TRUE)
        (print "debug information!")
    )
    ;Do something else
)
```

You can call this static script by using the simple command:

(do_something TRUE).

Command Scripts

Command scripts are scripts that will be individually executed by AIs. They are there to script the behavior of a particular squad member, so that he does something that he would not naturally do otherwise. Only within these scripts can you use most of the commands that start with **cs_**. You can attach a command script to a unit in sapien by putting the name of the command script in the "placement script" field of a squad or a squad's starting point. You can also use the following command to attach the example_script to squad_A:

```
(cs_run_command_script squad_A cs_example_script)
```

A very useful thing to prevent troops from looking ridiculous when under the influence of a command script is to make sure they react when they are injured, thus snapping out of their scripted behavior:

```
(cs_abort_on_damage true)
```

A monster under the influence of a command script will not shoot unless you explicitly give him the right to using the following commands:

```
(cs_enable_moving true)
(cs_enable_targeting true)
```

The probably most useful command for command scripts is cs_go_to, which explicitly tells the character to go to a specified point placed in Sapien. This will guide the monster executing the command script to the point p0 of point set pts_example_point_set:

```
(cs_go_to pts_example_point_set/p0)
```

How to exit a command script

There is three ways for a unit to exit a command script. The first way is the simplest: the unit finishes the command script, resuming whatever activity it had before. The two other ways are less trivial.

Ending with exit conditions

For each command script you can specify if you want the script to end in specific conditions. For example, to end the command script whenever the unit engages in combat, would use the following command:

```
(cs_abort_on_combat_status ai_combat_status_active)
```

There are also some other ai combat states that you can utilize. To end when the unit is on alert:

```
(cs_abort_on_alert TRUE)
```

To end when the unit receives damage:

```
(cs_abort_on_damage TRUE)
```

Ending by overriding with another command script

You can stop a unit from running a command script by telling the unit to run another command script that immediately ends.

```
(script command_script cs_exit
    (sleep 1)
)
```

To tell a unit to run the script above, you would use the following command:

```
(cs_run_command_script cov_elite_0 cs_exit)
```

Variables

Variables are really useful to keep track of things in a script. There are no local variables - variables that only exist within a certain scope - they are all global and are declared as follows:

```
(global boolean var_A)
```

The main types of variables you will need are:

boolean— TRUE or FALSE

short— Discrete numbers (0, 45, -590)

real— Floating points (0.2, -8.5647, 3.1416)

You can change the value of a global variable using the set command. The following example changes var_A to 3.

```
(set var_A 3)
```

Variables are also used by the game to change its behavior. These variables are usually booleans, but not always.

While playing the game you can for example at any time change the speed of the game by typing the following in console. The following will slow the game by a factor of 2.

```
    (set game_speed 0.5)
```

Simple Commands

Looking at existing scripts will probably be the most useful thing to do, but there are certain things that can be underlined upfront:

All commands begin and end with (parentheses)

A command always starts with the first parenthesis and the name of the command. Most functions then need parameters to follow. Ex:

- (save_game)
- (+ var_A 5)
- (print "debugging text to be seen ingame")
- In algebra commands, the following order is applied: (- 5 3) will return "2", and can be translated by **"5 – 3"**.
- Every script starts with the script block that has the startup tag in it: **(script startup mission)**
- ";" comments a line, and a text block within "; *"and "*;" will be commented as well

Print

Prints text to the screen while playing the game. Useful to debug scripts.

```
(print "Hello World")
```

Begin

The **begin** command is there only to make sure that multiple commands can be issued in places where only one command is expected. You will often use that command, and its use is really only to execute commands in a sequence. The next example prints two lines to the screen if var_A is equal to 3:

```
(if (= var_A 3)
    (begin
        (print "line 1")
        (print "line 2")
    )
)
```

If

Conditions are done with the if command.

```
(if (= var_A 3)
    (sleep 30)
    (sleep 60)
)
```

The above command will sleep 1 second if the global variable var_A is equal to 3, and sleep 2 seconds if it is different.

Sleep_Until

This command has multiple purposes, such as waiting until a certain condition is true, or waiting until certain other events have occurred. A simple command like the following will simply wait until either the squad

jackal_c_2 is entirely dead or until var_A is equal to 3:

```
(sleep_until
    (OR
        (= (ai_living_count jackal_c_2) 0)
        (= var_A 3)
    )
)
```

The condition is evaluated once every 30 ticks (once per second). The first condition evaluates to true if the number returned by the command ai_living_count is equal to 0.

The second way to use the command sleep_until is to create a looping sequence (the equivalent to a "while" in C++, java, etc.). The way to do this is by placing a begin statement right after the sleep_until begins, and by putting the exit condition as the last command of that sequence. The next example will print "Halo" every second, until variable var_A is equal to 3:

```
(sleep_until
    (begin
        (print "Halo")
        (sleep 30)
        (= var_A 3)
    )
)
```

There are two optional parameters for that command: the first one is the number of ticks between each evaluation (default is 30). The second argument is a timeout for the sleeping command. After the timeout is reached, the **sleep_until** command is skipped and the script goes on.

Advanced Commands

Checkpoints

Checkpoints should be placed regularly throughout your levels. The command to initiate a checkpoint is:

```
(game_save)
```

> **IMPORTANT**
> This command will not always save the game!

It will first wait for the player to be safe: the player is not engaged in combat, the player is not falling off a cliff, etc. If all these conditions are not met for a certain time (around 10 seconds), the command will terminate without saving the game. If you want the game to continue checking for the conditions to be met and save, use the following command:

```
(game_save_no_timeout)
```

Dialog

In game dialog consists of two types: scripted and non-scripted dialog. The non-scripted dialog is managed by the engine, so we won't describe it here. Scripted dialog is started when the designer wants it to. The following two commands will make Johnson speak, and then wait for him to finish his speech:

```
(sound_impulse_start sound\dialog\test\objective_sp_zanzibar\johnson_00 (ai_get_object mar_sgt_johnson) 1)
(sleep (sound_impulse_language_time sound\dialog\test\objective_sp_zanzibar\johnson_00))
```

The first function takes an object as a second argument. Notice how we converted from an AI squad to an object using the ai_get_object command. The second argument specifies the source of the dialog; it can be replaced by "none" so that the players always hear the dialog clearly (it is not coming from a particular object). The long path leading to johnson_00 actually refers to a sound tag.

Music

Starting and stopping music is really easy in scripts. Once a song is started, it will continue looping until you stop it, so be sure to do so. Below is an example of how to start and stop a piece of music within the game:

```
(sound_looping_start sound\music\ec_ingame\ec_ingame none 1)
(sound_looping_stop sound\music\ec_ingame\ec_ingame)
```

Character Animation

You can control the animation of a specific character in scripts. You usually do that for very specific moments, like a marine waving at you, Johnson pointing out an objective, etc. Here's a sample script for playing a custom animation:

```
(custom_animation (ai_get_unit ai_current_actor) "objects\characters\marine\marine" "combat:rifle:wave"
true)
(sleep (unit_get_custom_animation_time (ai_get_unit ai_current_actor)))
```

The first argument refers to the object that will execute the animation, while the second argument points in which tag to look for the animation. The third argument reflects the actual animation. These animations can be found by exploring the model_animation_graph tag in Guerilla. The second command waits for the animation to be finished before moving to the next point in the script.

Navigation Points

Activating and deactivating a waypoint is quite simple. You first need to have an object to attach the waypoint to, and you then activate/deactivate it with the following commands:

```
(hud_activate_team_nav_point_flag)
(hud_deactivate_team_nav_point_flag)
```

In this case the object is a scenery object named generator_E that was placed in Sapien.

Garbage collection

The engine can only manage a fixed number of objects in the environment. When a large number of enemies are killed in a small space, the amount of weapons and corpses can easily become too large. The engine then automatically garbage collects the objects: it removes weapons and corpses that are not in the same section as the players.

Although the engine manages how many corpses can stay in the environment at the same time, sometimes you need to manually tell it when to remove them. This can happen in certain situations where performance is an issue and the garbage collecting is not done often enough. The first way to force it to garbage collect is the safest: it tells the engine to do the usual garbage collecting now. The weapons / corpses that are in the non visible areas will disappear. The command looks like this:

```
(garbage_collect_now)
```

If that doesn't work, there is another, riskier, version:

```
(garbage_collect_unsafe)
```

This garbage collecting will delete every object in the environment, even those visible to the players. You should use it only when you control the position of the player and you know there are no weapons / corpses in that area.

AI Vision

Sometimes you need to play with the AI awareness to make sure certain events happen. You can first make sure a squad sees another one by using the following command:

```
(ai_magically_see marines cov_base_attackers)
```

The command above reveals the Covenant to the marines, even if they wouldn't normally see them. You can also prevent all squads from attacking a specific squad. This can be useful to prevent situations where the Covenant would shoot a Pelican instead of a marine, for example. In the following command, no one will attack the pelican (pelican_1):

```
(ai_disregard (ai_actors pelican_1) TRUE)
```

Randoms

Random Numbers

It is sometimes useful to get a random number, for purposes like waiting a random amount of time between each point in a patrol. A "random_range" command would look like this:

```
(sleep (random_range 15 30))
```

The above command causes the script to sleep for 15 to 30 ticks. Note that you can use random_range outside of a sleep command.

Random command execution

The replayability of a game is better when some encounters are randomized. You can randomize events with a simple command: begin_random. This is a normal begin command, with the difference that it will execute all tasks in a random order. You can still have sequential events occurring randomly using nested begin commands, like in the following example:

```
(begin_random
    (begin
            (cs_go_to pts_cov_base_attackers/p0)
            (sleep_until (= var_flag_0 TRUE) 30 210)
    )
    (begin
            (cs_go_to pts_cov_base_attackers/p1)
            (sleep_until (= var_flag_1 TRUE) 30 210)
    )
)
```

In the above example a unit will first go to either p0 or p1, wait for a flag to be true, and then go to the other point he didn't explore yet. There is no easy way to have only one command execute randomly, but you can accomplish it using a global variable. The following example will print either A, B, or C:

```
(global boolean var_cmd_done FALSE)
(begin_random
    (if (= var_cmd_done FALSE)
            (begin
                (print "A")
                (set var_cmd_done TRUE)
            )
    )
    (if (= var_cmd_done FALSE)
            (begin
                (print "B")
                (set var_cmd_done TRUE)
            )
    )
    (if (= var_cmd_done FALSE)
            (begin
                (print "C")
                (set var_cmd_done TRUE)
            )
    )
)
```

Object Management

All objects are placed in Sapien, but sometimes you want to control when they appear on the map. To prevent an object from being created when the map is loaded, you must set the flag "not automatically" to true in Sapien. You then place or delete the object in script using the following commands:

```
(object_create weapons_BR_0)
(object_destroy weapons_BR_0)
```

You can also poll an object to get its health. The sample below will return a real number between 0 and 1:

```
(object_get_health generator_D)
```

Contextual Examples

Tracking the player's position

It is a good practice to always have a separate thread testing your trigger volumes. This ensures you don't have two different threads testing the same trigger volume. You can either just latch the player's position to TRUE, or put it back to FALSE when the player is not in the space anymore (if that is important for your needs). Here is a sample script with the two examples in it:

```
(global boolean var_pos_room_a_0 0)
(global boolean var_pos_room_a_1 0)
```

```
;*Update global variables indicating what is the player's current position*;
(script dormant room_a_pos_updater
    (sleep_until
            (begin
                (if (= (volume_test_objects vol_room_a_0 (players)) TRUE)
                        (set var_pos_room_a_0 1)
                        (set var_pos_room_a_0 0)
                )
                (if (AND
                        (var_pos_room_a_1 FALSE)
                        (= (volume_test_objects vol_room_a_1 (players)) TRUE)
                )
                (set var_pos_room_a_1 1)
                )
            )
    )
)
```

Guiding the marines (or other player allies)

You probably want a separate thread (dormant script) to lead the marines within the level. The reason behind this is simple: the marines will sometimes wait on different conditions than the monsters to move. If a single script is used, you might end up waiting for a walking condition for the monsters while the marines should have moved. When both the marines and the monsters are waiting for the same condition, you can create a trigger within Sapien that can be queried from both scripts using the command **ai_trigger_test**.

Patrolling

There are two ways to do a patrol. You can give the squad a bunch of firing points and have an order on "idle" combat status. The squads under that order will randomly go from a firing point to another. If you want more control, you need to use a command script that precisely tells where to go next, like the example below:

```
(script command_script cs_A_elite_patrol_0
    (cs_abort_on_combat_status ai_combat_status_active)
```

Above is the exit condition that allows you to stop patrolling whenever the squad engages the enemy.

```
    (cs_enable_pathfinding_failsafe true)
```

The default behavior for cs_go_to commands is to skip the walking point if a small pathfinding problem occurs. You want to turn that variable to true to make sure the AI will really go to the specified points.

```
    (sleep_until
            (begin
                (cs_go_to pts_A_elite_patrol_0/p0 1)
```

The real command that makes it all possible. pts_A_elite_patrol_0/p0 is a point that is part of the pts_A_elite_patrol_0 point set. They are placed in Sapien.

```
                (cs_go_to pts_A_elite_patrol_0/p1 1)
                (sleep (random_range 15 30))
```

We usually make sure to add some amount of sleeping between two points, so that the unit stops for while, looks around, and then continues his patrol.

```
                (cs_go_to pts_A_elite_patrol_0/p2 1)
                (sleep (random_range 15 30))
                (cs_go_to pts_A_elite_patrol_0/p3 1)
                (sleep (random_range 60 90))
    false)
```

Always loop: patrolling goes back to the first point.

```
    )

)
```

Dropships

Dropships are all scripted. You tell them where to go, how to get there, at which speed, etc. Let's look at a script example.

Loading the guys in - This function loads in guys you would have spawned just before. They will instantly teleport to their respective seat.

```
      (vehicle_load_magic (ai_vehicle_get_from_starting_location LZ_phantom_02/pilot) "phantom_p" (ai_actors
LZ_jackals_phantom_02))
```

The ride

```
      (script command_script LZ_phantom_arrives_01
            (cs_enable_pathfinding_failsafe TRUE)
            (cs_vehicle_speed .5)
            (cs_vehicle_boost TRUE)
```

The speed of the vehicle from 0 to 1, and if you want the phantom to come in really fast (boost).

```
            (cs_fly_by LZ_airspace/ph0a)
```

cs_fly_by means that the phantom will pass near the point but will not stop at it.

```
            (cs_vehicle_boost FALSE)
            (cs_fly_by LZ_airspace/ph1a)
            (cs_fly_by LZ_airspace/ph2a)
            (cs_vehicle_speed .3)
            (cs_fly_to LZ_airspace/pel6 2)
```

This time the phantom will stop at this point before moving on to the next command.

```
            (cs_fly_to_and_face LZ_airspace/pel6 LZ_airspace/p0 1)
```

Same as cs_fly_to, but the phantom position itself to face the second point given in argument.

```
            (wake lz_phantom_01_drop)
```

You usually wake a different thread that will drop the covenants on that spot. Waking a different thread to drop while still moving is an option. The other option is to wait for the phantom to unload before moving.

```
        (cs_fly_to_and_face LZ_airspace/ph3a LZ_airspace/p0 1)
        (sleep_until (<(ai_living_count lz_phantom_01) 2) 30 900)
```

You want to wait until a certain condition before taking off. Sleep at least long enough to let your guys drop off the phantom.

```
        (cs_fly_by LZ_airspace/ph2a)
        (cs_vehicle_speed .5)
        (cs_fly_by LZ_airspace/ph1a)
        (cs_fly_by LZ_airspace/ph0a)
        (cs_fly_to LZ_airspace/phxa)
        (ai_erase LZ_phantom_01)
```

You always want to erase a guy once he's out of sight. Make sure to lead the phantom far enough so that you don't hear its ambient sound no longer.

```
    )
```

Dropping the guys - Dropping guys is straight forward. The only thing you need to remember is to have some time between drops so the squads don't land on each other's head.

```
    (script dormant lz_phantom_02_drop

        (object_set_phantom_power (ai_vehicle_get_from_starting_location lz_phantom_02/pilot) TRUE)
```

You want to activate power on a phantom so that the dropping bay opens and that the phantom can actually drop its troops.

```
        (vehicle_unload (ai_vehicle_get_from_starting_location lz_phantom_02/pilot) "phantom_p_a01")
```

This function unloads troops from the phantom_p_a01 seat. There could be guys in the following seats, as well as in the phantom_p_b... and phantom_p_c... seats. See the phantom tag in guerilla to get a list of all the seats.

```
        (sleep 15)
        (vehicle_unload (ai_vehicle_get_from_starting_location lz_phantom_02/pilot) "phantom_p_a02")
        (sleep 15)
        (vehicle_unload (ai_vehicle_get_from_starting_location lz_phantom_02/pilot) "phantom_p_a03")
        (vehicle_unload (ai_vehicle_get_from_starting_location bridge_phantom_01/pilot) "phantom_lc")
```

This will unload the large cargo, which usually contains a ghost.

```
        (sleep 60)
        (object_set_phantom_power (ai_vehicle_get_from_starting_location lz_phantom_01/pilot) FALSE)
    )
```

Doors & Switches

The way doors and switches connect is done through scripting. You have to always check for the position of a switch, and when that position changes you decide to do what you want next, usually open a door. This can be done really easily with the following two commands:

```
    (sleep_until (> (device_get_position beach_switch) 0))
```

device_get_position returns the position of the switch, which is 0 by default. When the player activates it, it changes from 0 to 1. The switch is a device control object named beach_switch.

```
(device_set_position beach_door 1.0)
```

This command simply opens a door, beach_door, a device machine object.

AI/Vehicle Interaction

Here is some information about AIs and vehicles.

Enter/Exit

Two quick commands to know: **ai_vehicle_enter** and **ai_vehicle_exit**. Here's an example on how to use them:

```
(ai_vehicle_enter mar_sgt_johnson (ai_vehicle_get_from_starting_location warthog_0/driver)
"warthog_d")
```

The command above tells the mar_sgt_johnson squad to try to get into the vehicle that was originally spawned by the warthog_0 squad, in the driver position. The seat is specified with the last argument (warthog_d). The following command tells anybody occupying the warthog to exit the vehicle. The warthog is referenced as a vehicle that was originally spawned by the warthog_0 squad:

```
(ai_vehicle_exit warthog_0)
```

Some sandbox behaviors may override these commands. For example, if a unit is driving a vehicle and the player makes him exit (by pressing X), the AI driver won't enter the vehicle again for a long time. Even the ai_vehicle_enter command won't for him to.

Reserving Seats

As part of their sandbox behavior, brutes, elites and even grunts will use vehicles by themselves. Sometimes you want to prevent that situation by making a vehicle impossible to drive. This can be achieved by reserving a vehicle, or a specific seat.

```
(ai_vehicle_reserve_seat (ai_vehicle_get_from_starting_location warthog_0/driver) "warthog_d" TRUE)
```

In the above command we tell every AI not to drive the vehicle that was spawned with the warthog_0 squad. Sometimes you want to prevent the player from driving a vehicle (maybe it wasn't introduced as a drivable vehicle yet). This can be done using the following command:

```
(unit_set_enterable_by_player (ai_vehicle_get_from_starting_location warthog_0/driver) FALSE)
```

Driving a Vehicle

You can let an AI drive a vehicle by himself, giving firing points from a vehicle area. You can also script a ride so that the driver always takes the same route. This is done quite easily with points placed in Sapien, in a point set.

```
(script command_script cs_warthog_escape
      (cs_enable_pathfinding_failsafe true)
      (cs_vehicle_speed 0.75)
      (cs_go_to pts_marines_warthog/p0 2)
```

The line above tells the AI to drive to a specific point. The second argument, 2, is the acceptable radius around the point. If the vehicle is within that range, the cs_go_to command is considered accomplished.

```
        (cs_go_to pts_marines_warthog/p1 2)
    )
```

After this command script has ended, the AI driver will look for his normal firing points and use them.

Effects

Playing from Scenery objects

You use scenery objects when you want an effect that is permanent. You place a scenery effect (usually under tags/effects/scenery) and deactivate it with the "not automatically" flag. You can then activate it with the following command:

```
    (object_create_anew fx_0)
```

The above command creates the effect scenery object named fx_0.

Using flags

This effect is used for quick scripted explosions, sparkles, etc. You first need to place a flag in Sapien. You then choose an effect you like. An effect can include sound, visual effect, damage, etc:

```
    (effect_new "effects\scenarios\solo\alphagasgiant\wall_explosion" flag_fx_0)
```

The command above will start an explosion on the flag flag_fx_0.

** Volume Return Objects by Type

Below is information concerning the usage of the volume_return_objects_by_type HS command.

Here's the enumeration that's used:

```
enum e_object_type

{

_object_type_biped,

_object_type_vehicle,

_object_type_weapon,

_object_type_equipment,

_object_type_terminal,

_object_type_projectile,

_object_type_scenery,

_object_type_machine,

_object_type_control,

_object_type_light_fixture,

_object_type_sound_scenery,

_object_type_crate,

_object_type_creature,

_object_type_giant,

k_object_types_count,

_object_type_first= _object_type_biped,

_object_type_none= NONE

};
```

The number that corresponds to each entry is 2^entry_index (where entry_index starts at 0). So biped is 2^0= 1 and projectile is 2^5= 32.

You can add numbers together to get masks: 35= 32 + 2 + 1= 2^5 + 2^1 + 2^0, so that would mask projectile, vehicle, and biped.

Helpful Hints

This section is just a bunch of points that one might struggle with when learning the scripting language.

- Some commands need an "object" type parameter. When you want to use that command on a squad, you need to first convert it using the command ai_actors.
- You might find it useful in some command scripts to have a reference to the unit that is currently executing it. The unit currently running the script can be accessed with ai_current_actor.

Guerilla

11/23/2022 • 2 minutes to read

Guerilla is the main interface for editing tags.

User Interface

Information on Guerilla's User Interface.

Edit Menu

Information on Guerilla's Edit Menu.

File Menu

Information on Guerilla's File Menu.

Filters

Information on Guerilla's Filters.

Source Control Menu

Information on Guerilla's Source Control Menu.

View Menu

Information on Guerilla's View Menu.

Window Menu

Information on Guerilla's Color Picker.

Color Picker

Information on Guerilla's Edit Menu.

Help Menu

Information on Guerilla's Help Menu.

User Interface

11/23/2022 • 2 minutes to read

Upon launching Guerilla for the first time, the interface should appear similar to (see Figure 1). Along the left side of the window is a frame containing a list of folders, and on the right is a large empty area — this is where you will work with any open tags. It's important to note that the file tree on the left is a graphical representation of any tags files you have located at \halo3\main\tags on your hard drive. Guerilla always (and only) points to that specific location.

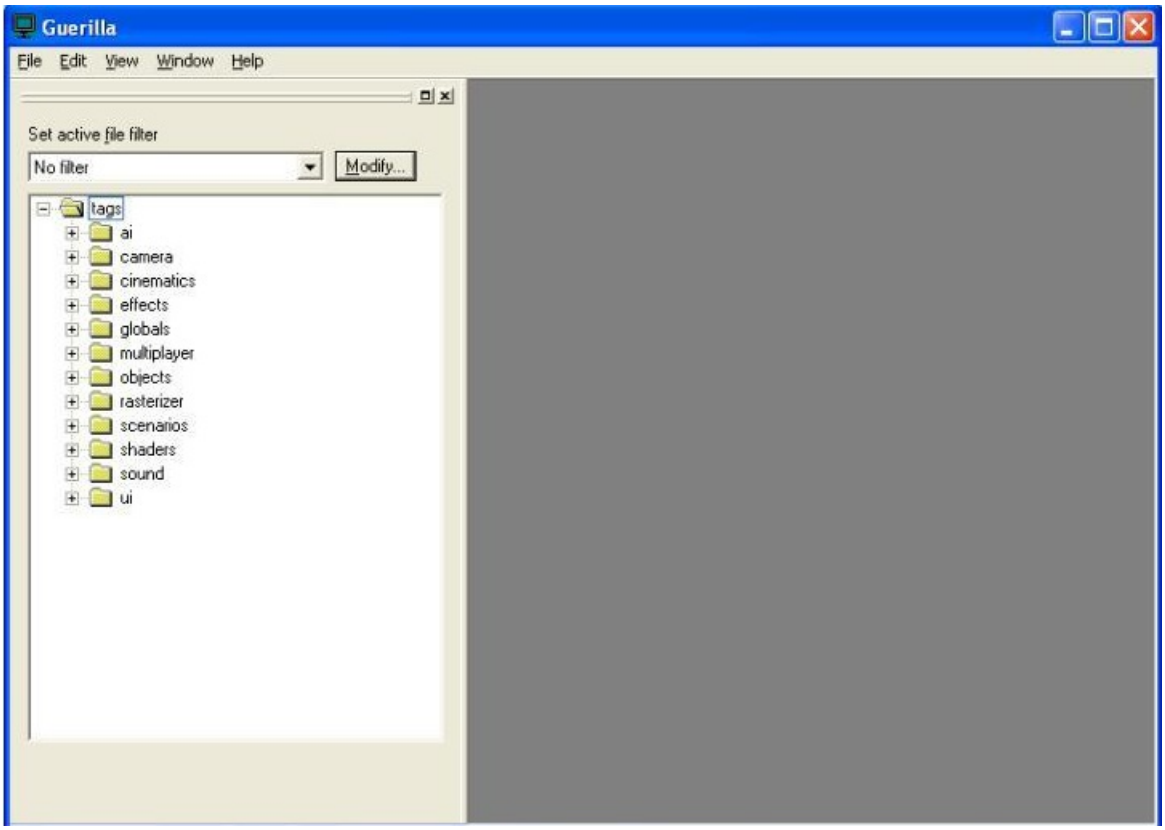


Figure 1 - The main Guerilla interface

Edit Menu

11/23/2022 • 2 minutes to read

The Edit menu is similar to the standard edit menu that appears in most windows programs. There are the common commands like undo, cut, copy, and paste— which all allow you to easily manipulate text strings— along with some other commands which are unique to Guerilla.

> **NOTE**
> Some of the items on this menu only appear when a tag is opened for editing.

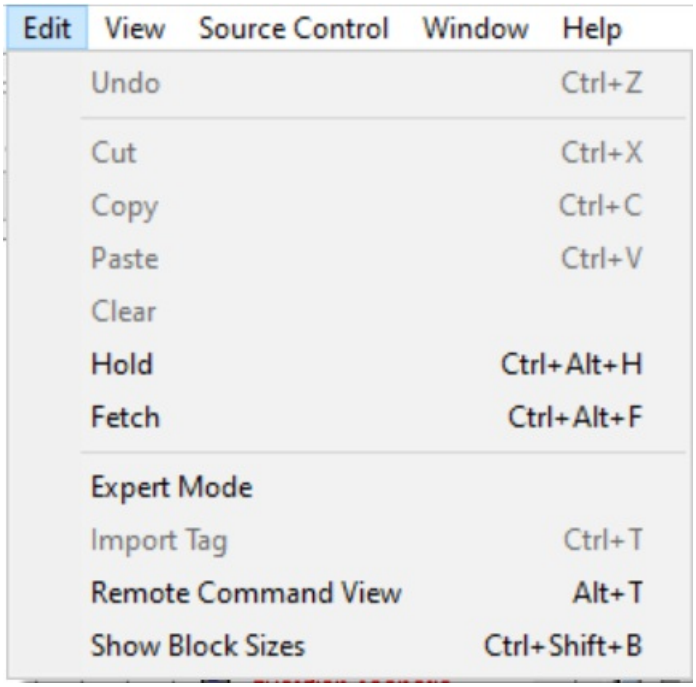


Figure 1 - Guerilla's Edit Menu

- **Undo**— Undoes the last change you made to a particular tag.
- **Cut**— When an item (block of text, image, etc) is selected, this functions deletes it from the tag and moves it to the Windows clipboard (for use in pasting elsewhere).
- **Copy**— When an item (block of text, image, etc) is selected, this function makes a copy of it and places the copy on the Windows clipboard (for use in pasting elsewhere).
- **Paste**— Pastes a previously copied/cut piece of data.
- **Clear**— Clears (deletes) the selected piece of data.
- **Hold**— Saves the current state of the open tag without permanently saving it. You can "hold" a tag, Xsync, and view your changes in the game engine without having to save the tag. Then, if you don't like the changes, you can select "Fetch" and restore the tag to its previous state.
- **Fetch**— Restores a tag with a "hold" on it to the state it was in when the hold was set.
- **Expert Mode**— Allows access to some sections of tags that are normally not editable. You should not be using this feature if you don't already know what it does!
- **Import Tag**— Uses tool command to import a tag into Guerilla.
- **Remote Command View**— Legacy option, no current functionality.
- **Show Block Sizes**— When this option is selected (checkmark appears next to it in the menu), the file size of all blocks in a tag appear next to the block titles.

File Menu

11/23/2022 • 2 minutes to read

The File menu (see Figure 1) provides many of the commands you'll need to interact with tags, create tags, and import your work into the game engine. Below is a brief description of each of the commands on the file menu and what its functions are.

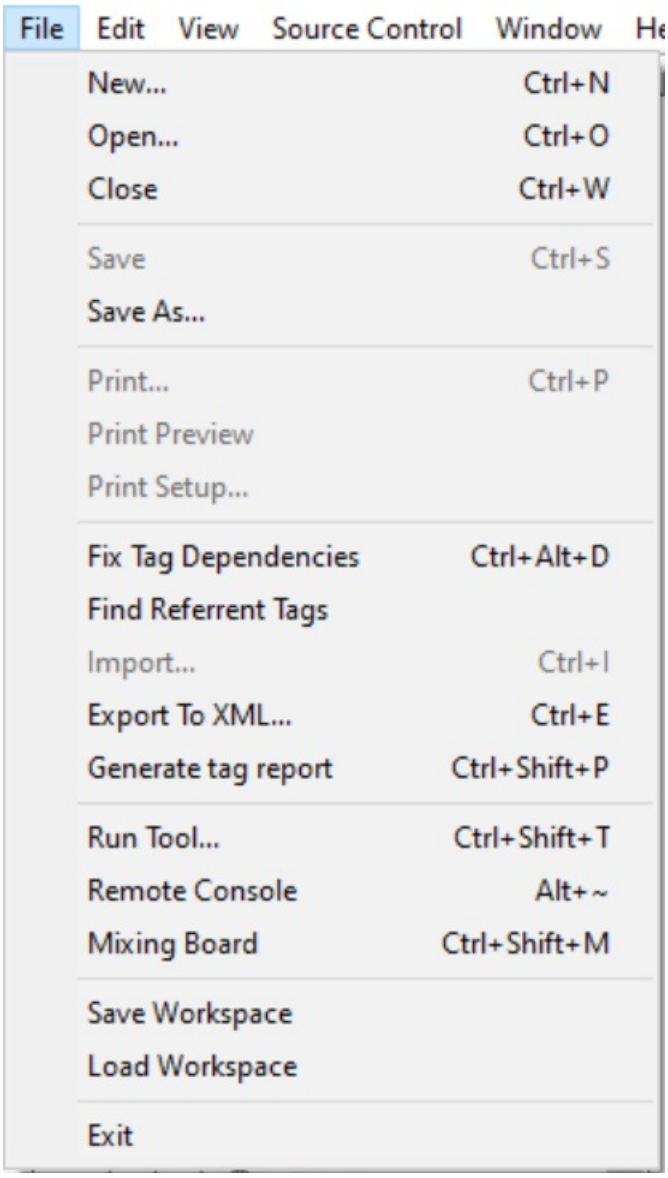


Figure 1 - Guerilla's File Menu

- **New** (Ctrl+N)— Opens the dialog for creating a new Tag.
- **Open** (Ctrl+O)— Brings up a chooser which lets you select a tag file to open and edit.
- **Close** (Ctrl+W)— Closes the currently opened and selected tag.
- **Save** (Ctrl+S)— Saves the tag which you are currently editing.
- **Save As**— Brings up a dialog which allows you to choose the name and location at which to save the tag you are currently editing.
- **Fix Tag Dependencies** (Ctrl+Alt+D)— Selecting this option builds a database of all tag references, checks to see which tags refer to a tag that doesn't exist, and then generates a report. From there, the user is given the option to remove the bad references or pick a new tag for those tags to refer to. For more information on

this command, check out the *Tag Dependency Information* article.

- **Find Referrent Tags**— Generates a list of all tags which refer to the currently opened tag. Brings up a dialog box (see Figure 2) which displays the names and locations of the referring tags and gives the user the opportunity to open them. If you haven't yet generated a tag dependency database, running this command will do that for you. However, this could take a very long time, so be prepared to be unable to use your machine for the next half hour (give or take 10 minutes). See the *Tag Dependency Information* article for details on tag dependencies.



Figure 2 - The Find Referrent Tags Dialog.

- **Import** (Ctrl+I)— Imports a correctly formatted text file into guerilla as a tag.
- **Export** (Ctrl+E)— Exports all information in a tag to a text file.
- **Generate Tag Report** (Ctrl+Shift+P)— Generates a list of all debug output a tag will generate when run in the game engine.
- **Run Tool** (Ctrl+Shift+T)— Runs the program Tool.exe from within Guerilla. This command creates a dialog box (see Figure 3) with a drop down menu of all Tool commands. It is useful for (among other things) importing content created in a 3rd party program — such as Max or Maya — into the game engine.

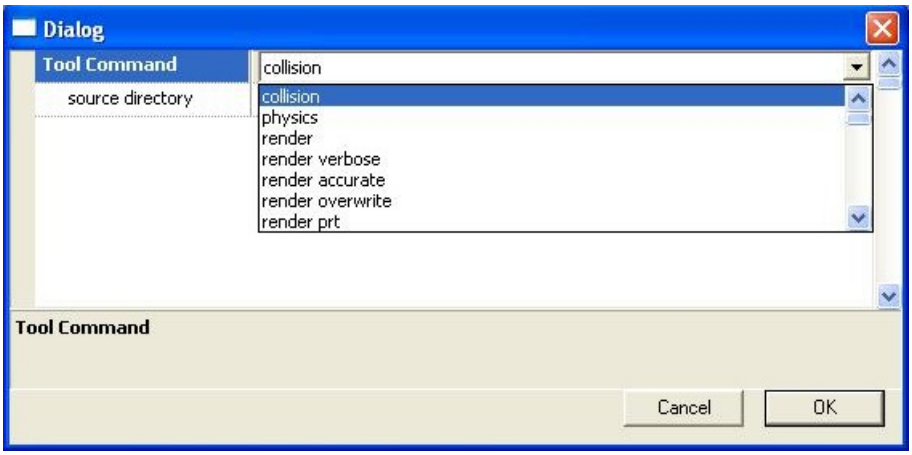


Figure 3 - The Run Tool command dialog

- **Remote Console** (Alt+~)— Launches a telnet console connected to your currently connected Xbox. Allows you to run console commands via your PC rather than using a keyboard with your Xbox.
- **Mixing Board** (Ctrl+Shift+M)— Brings up a dialog which allows the user to launch either the sound or cinematic mixing board.
- **Save Workspace**— Allows you to save the layout and settings of your current Guerilla workspace. Your default workspace is saved at (PATH-TO-H3EK-DIRECTORY)\prefs\guerilla.cfg. This would be a good location to save any other workspace layouts.
- **Load Workspace**— Loads a previously saved custom workspace.
- **Exit**— Exits and closes Guerilla.

Filters

11/23/2022 • 2 minutes to read

Directly above the file tree in the left frame of the Guerilla window, is a drop down menu with text above it that reads "Set active file filter" (see Figure 1). Filtering allows the user to select and view only a specific set or type of tags— thus "filtering" out any tags you may not need or want to see.



Figure 1 - The Filter drop down menu.

To set up a filter

1. Click the **Modify** button next to the drop down list. This will bring up the Modify Filters dialog box (see Figure 2).

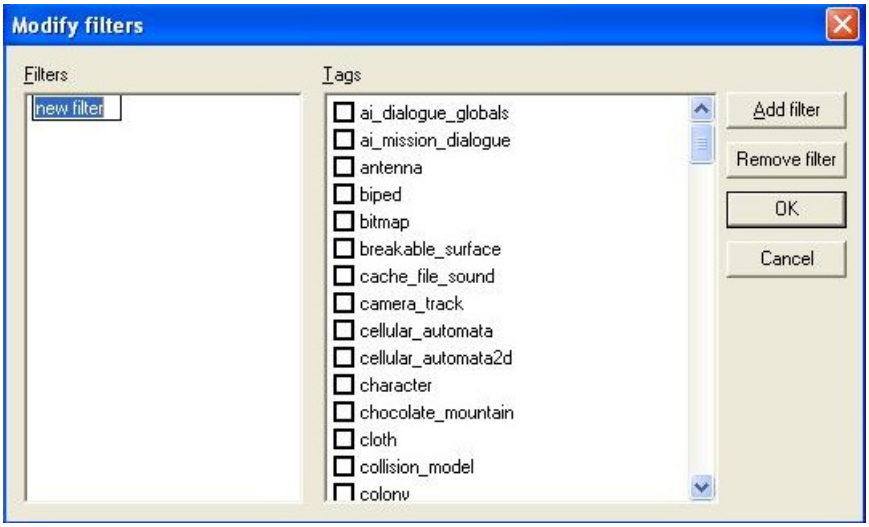


Figure 2 - The Modify Filters Dialog

2. Click the **Add Filter** button. This will add an item in the Filters frame on the left called "New Filter." As you can see, the name of the filter is selected and can be changed to a name specific to the filter being created.
3. Check boxes in the Tags frame on the right for any tags you want to see displayed.
4. When you're finished selecting the tags to be displayed, click **OK**.

Once you've added a filter, it should appear in the drop down list in the main window. When you have the filter selected, you will still see all of the tags folders, but you will only be able to see tags of the type you selected when creating the filter. Any filters you create are stored in the (PATH-TO-H3EK-DIRECTORY)\prefs folder in a file

called tag_filters.xml.

Source Control

11/23/2022 • 2 minutes to read

The Guerilla Source Control menu (see Figure 1) has various options for source control.

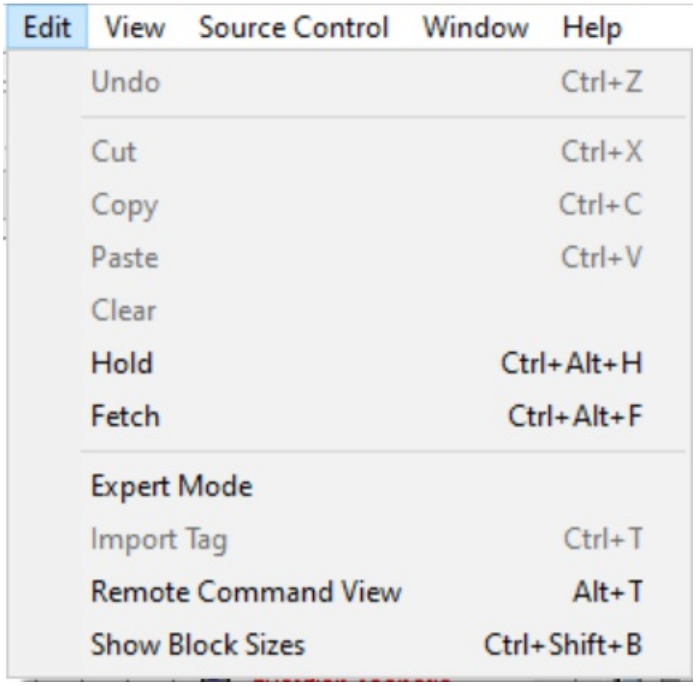


Figure 1 - Source Control Menu

- **Get Latest (Ctrl+G)**— Gets latest from Source Depot.
- **Sync & Check Out (Ctrl+K)**— Syncs and checks out the content.
- **Check In (Ctrl+U)**— Checks in the content.
- **Undo Check Out**— Undoes the checkout procedure for the item.
- **Show History**— Shows the history of what's been checked in and out.
- **Make Writeable**— Makes the item writable.

View Menu

11/23/2022 • 2 minutes to read

Guerilla has a very simple layout, so the view menu (see Figure 1) is also simple. There are only two options: Explorer Bar, and Output.



Figure 1 - Guerilla's View Menu

- **Explorer Bar** — Toggles the display of the file tree frame (the explorer bar)
- **Output** — If you need to know what this does, talk to Mat. Otherwise, don't worry about it

Window Menu

11/23/2022 • 2 minutes to read

The Window menu (see Figure 1) controls the way your open tags are displayed and gives you direct access to bringing focus to a specific open tag.

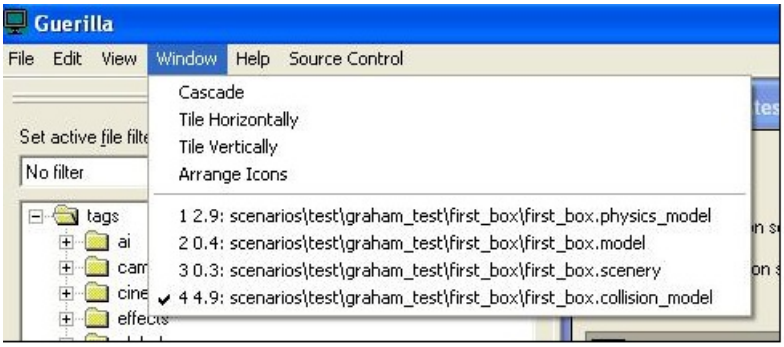


Figure 1 - View of the Window Menu

- **Cascade**— Arranges any open tag windows in a diagonally cascading fashion.
- **Tile Horizontally**— Any open tag windows are tiled in rows down the screen (does not work when you have 1, 4, or 5 open tags).
- **Tile Vertically**— Any open tag windows are tiled in columns down the screen (does not work when you have 1, 4, or 5 open tags).
- **Arrange Icons**— Arranges all minimized tag windows.
- **Window Selection**— The lower area of the menu lists all of the tags that are currently opened for editing/viewing. You can select any of the names on the list (or type the hotkey number) to bring that window into focus.

Color Picker

11/23/2022 • 2 minutes to read

The Color Picker available in Guerilla is similar to the one you might find in Photoshop. You can define the color you want by entering a numeric value in either HSB or RGB, or you can point and click to get the color you want. The Color Picker also allows you to save and load custom palettes (similar to the way color swatches work in Photoshop).



Figure 1 - The Color Picker

How it Works

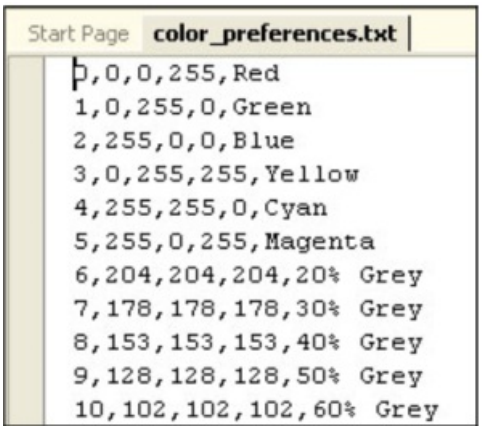


Figure 2 - the color_preferences.txt file format

All of the default color information for the swatches in the color palette is stored in a text file called **color_preferences.txt** which is located in **main\prefs**. There are five columns in the color_preferences file (see Figure 2):

- Swatch #— Each swatch is numbered (starting at 0) from left to right, top to bottom
- Red Value
- Green Value
- Blue Value
- Color Name/Comment— Each swatch can have a name/comment that appears when the user mouses over it

You can open the color_preferences.txt file in a text editor (such as Notepad, Visual Studio, etc.) and edit the color values and names to suit your needs, but creating a custom file (see below) would be better since the default

color_preferences file gets updated by Source Depot.

Each time the Color Picker is closed, the color_preferences.txt file is automatically saved, so it remembers which colors you were working with previously (until you close and re-open Guerilla). The same is true for any custom color palettes you load into the color picker (see below for instructions).

Create and Save a Custom Palette

You can create and save a custom color palette (or many different color palettes) by following these steps:

1. Open the Color Picker. The default values of the color preferences will be loaded into the palette.
2. Adjust the RGB or HSB values to create a color that you like (or drag the slider and click in the color window).
3. Click in the currently selected color swatch and drag it down to any swatch on the color palette (see Figure 3).

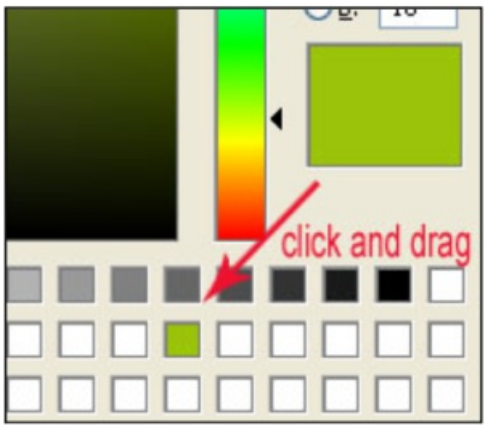


Figure 3 - Click and Drag to add a color to the palette

4. Repeat with as many colors as you want to add to your custom palette. Remember, you can always drag one color over the top of another to replace it.
5. Click the **Save Palette** button in the bottom of the color picker window.
6. Navigate to the location you'd like to save your file, give it a memorable name, then click **Save**.

Load a Custom Palette

You can load a previously saved custom color palette into the color picker by doing the following:

1. Open the Color Picker in Guerilla.
2. Click the **Load Palette** button.
3. Navigate to your custom palette text file, click to select it, and then click the **Open** button. Your palette should be loaded into the picker.

Help Menu

11/23/2022 • 2 minutes to read

The Help menu in Guerilla only contains one item: **About Guerilla**.

Selecting About Guerilla brings up a splash screen with information about the program (see Figure 1).

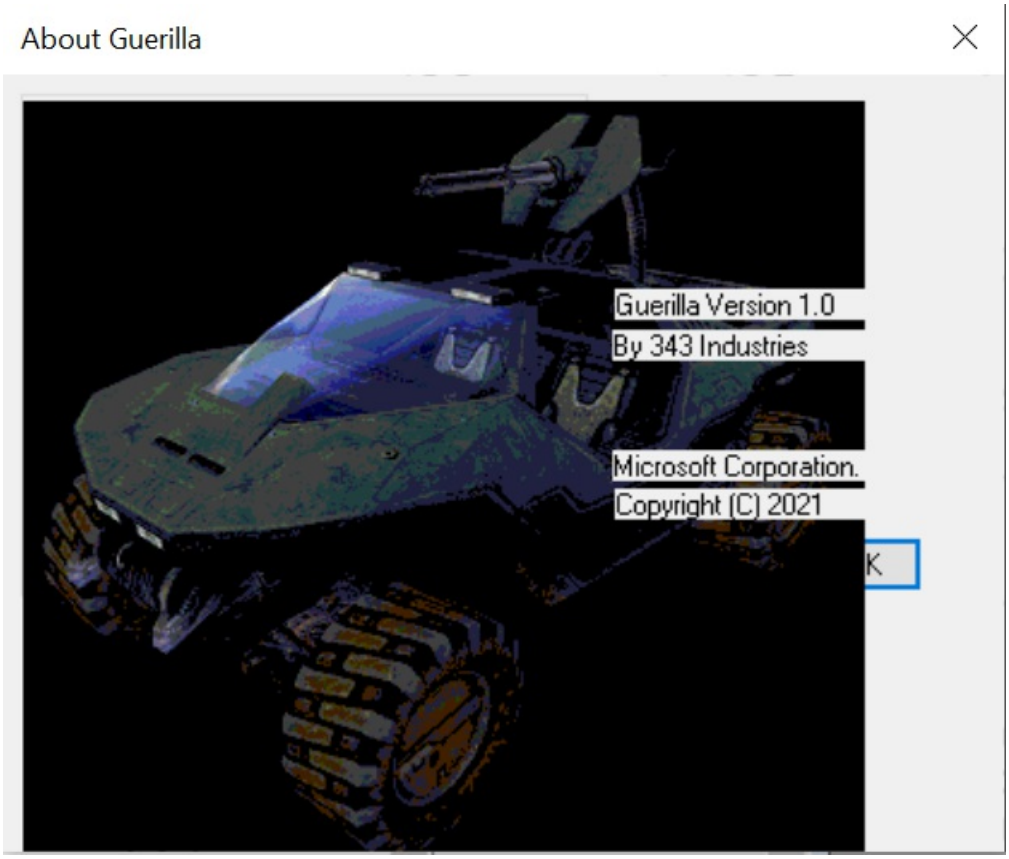


Figure 1 - Guerilla splash screen

Sapien

11/23/2022 • 2 minutes to read

Sapien is the main interface for editing a scenario (MP and Campaign maps or Cinematic BSPs). This includes placing objects, decals, lighting, characters, vehicles, and setting up encounters with friendly and enemy AI.

Overview

A general overview of the Sapien tool.

Game Window

Information on Sapien's Game Window.

Hierarchy View

Information on Sapien's Hierarchy View.

Tool Window

Information on Sapien's Tool Window.

Asset Manipulation Gizmo

Information on Sapien's Asset Manipulation Gizmo.

Output Window

Information on Sapien's Output Window.

Placing Lights

Information on how to place objects in Sapien.

Placing Objects

Information on how to place lights in Sapien.

Properties Palette

Information on Sapien's Properties Palette.

Structure Paintier

Information on Sapien's Structure Painter.

Sapien Overview

11/23/2022 • 6 minutes to read

Sapien is our main interface for editing a scenario (MP and Campaign maps or Cinematic BSPs). This includes placing objects, decals, lighting, characters, vehicles, and setting up encounters with friendly and enemy AI.

For detailed information about Sapien's various features, visit these articles:

User Interface

Menus

- **File** (see Figure 1)— Provides access to the following commands:
 - **New**— Creates new file.
 - **Open Scenario** (Ctrl+O)— Brings up an open dialog, which allows you to open a scenario. Opening a different scenario automatically closes the currently open one.
 - **Save Scenario** (Ctrl+S)— Saves your scenario.
 - **Save Scenario As**— Brings up a dialog which allows you to re-name your scenario and/or save your scenario to a different location.
 - **Compile Scripts** (Ctrl+Shift+C)— Compiles any scripts you have written in HaloScript to attach to the current scenario.
 - **Export Script Names** (Ctrl+Shift+E)— Exports script names associated with the scenario to a file called hsc_scenario.syninc (places it in main folder you've mapped your depot to).
 - **Xbox Sync** (Ctrl+Shift+S)— Syncs all of the files in your \tags... directories to your Xbox.
 - **XSync Scenario** (Ctrl+Shift+F)— Syncs only the currently open scenario to your Xbox.
 - **Connect to Xbox** (Ctrl+Shift+X)— Brings up a dialog which allows you to enter the name or IP address of an Xbox to connect to in order to sync files or run commands.
 - **Reboot Xbox** (Ctrl+Shift+R)— Reboots the Xbox you are currently connected to.
 - **Reboot Xbox to Game** (Ctrl+Alt+R)— Reboots the Xbox you are currently connected to and launches the tagdebug.xbe (executable).

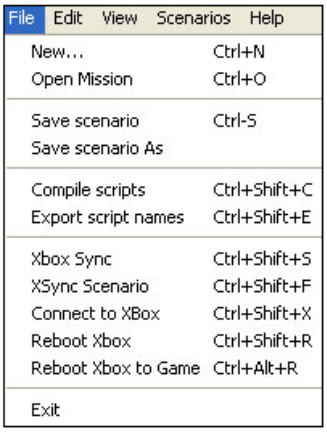


Figure 1 - Sapien's File Menu

- **Edit** (see Figure 2)— Provides access to the following commands:
 - **Switch BSP** (Ctrl+B)— Scenarios can contain multiple BSP's (Binary Space Partitions). This command brings up a dialog (Figure 3) with a drop down list which allows you to switch to a different BSP in the Game Viewer.
 - **Drop Objects** (Ctrl+D)— Currently non-functional.

- **Expert mode** (Ctrl+Alt+Shift+X)— unlocks use of some fields in the properties palette that are otherwise unusable. Don't turn this on unless you know what you're doing, or someone else has told you exactly what to do with it.
- **Reset Object Z** (Ctrl+Alt+Z)— Resets the object's Z-plane coordinate to zero. You can set a maximum distance that the reset command will work within. For example, you can set the Max Distance property to 3 and if your object is greater than 3 WU away from 0 on the z axis, Reset Z will ignore the object (this is handy when selecting and moving multiple objects). You can also choose to only move objects either up or down along the z-axis to zero. So, if you select up and your object is above 0 on the z-axis, your object won't be moved.
- **Copy Object Transform** (Ctrl+K)— Copies the exact position and orientation of a selected object within your scenario to the clipboard.
- **Apply Object Transform** (Ctrl+L)— Pastes a selected object to the exact position and orientation of a previously copied object from the clipboard. Be careful with this one: it pastes the object exactly over the top of the copied object, so in order to see your pasted object, you'll have to select it and move it.
- **Hexidecimal Mode**— Changes some numbers (such as unique ID's for objects) to hexidecimal format.
- **Clear Output Window**— Clears the Output Window of any messages/error logs.

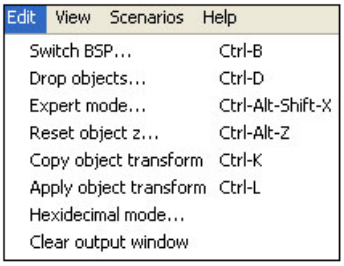


Figure 2 - Sapien's Edit Menu

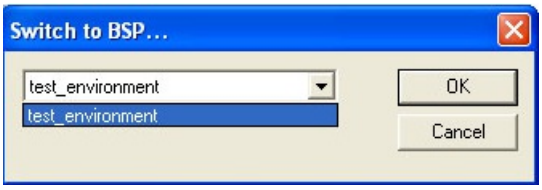


Figure 3 - The Switch BSP Dialog



Figure 4 - The Reset Object Z Dialog

- **View** (see Figure 5)— Provides access to the following commands:
 - **Toolbar**— Toggles the toolbar on/off (checkmark next to the command signifies on).
 - **Status Bar**— Toggles the Status Bar (the bar that runs along the bottom of the sapien window and provides helpful tips about commands) on or off.
 - **Game View**— Toggles the Game View window on and off.
 - **Property View**— Toggles the Properties Palette window on or off.
 - **Hierarchy View**— Toggles the Hierarchy View window on and off.
 - **Tool View**— Toggles the Tool Window on and off.
 - **Reset Window Prefs**— Resets the Sapien Windows settings to the default settings. The window

arrangement you have upon closing sapien will be saved in your preferences.



Figure 5 - Sapien's Edit Menu

- **Scenarios** (see Figure 6)— Provides access to the following commands:
 - **Run Game Scripts** (Alt+G)— Attempts to run any scripts you have included with the scenario.
 - **Map Reset** (Alt+R)— Resets the map to the state it is in upon startup/launch.
 - **Generate all Pathfinding Data**— Creates pathfinding data for all of the AI placed in the scenario. This command needs to be run before AI will be able to move.
 - **Split Mission Resources** (Ctrl+Shift+P)— Splits up the scenario into various resource tags, which make it possible for multiple people to work on different aspects of the scenario at the same time. Running this command creates a Resource directory in the root folder of your scenario and places the new resource tags in it. Once the command is run, opening the .scenario tag will still show the whole scenario but you'll need to check out individual resource tags in order to edit them.
 - **Split Mission Scripts**— Performs the same function as Split Mission Resources, but instead of splitting resources, it splits scripts out of the .scenario tag and gives them their own scripts directory in the root folder of your scenario (in \tags). IMPORTANT: After you run Split Mission Scripts, you won't be able to add new scripts to the scenario by simply saving them in the correct location in \data\ and compiling. Because (after splitting) the scripts are now in their own resource tags, you need to run Add Mission Script to create a new script tag, then compile the script to get it to run correctly in your scenario.
 - **Add Mission Script**— Only available after running Split Mission Scripts. Brings up a dialog which allows you to choose a script, then it creates a .scenario_hs_source_file in your \tags\scripts directory. You will still need to compile scripts to get the new script to run in your scenario. After you've run split mission scripts, this is the only way to add a script to your scenario.
 - **Split Mission AI**— Like the Split Mission Scripts and Split Mission Resources commands, Split Mission AI separates the AI tags from the .scenario tag and places them in two categories: cinematic and mission. Again, this is mainly so that multiple people can work on the AI at the same time.
 - **Set Active Mission AI**— After you've run Split Mission AI, this command is used to set which AI are part of the mission AI and which are part of cinematic AI.
 - **Import Comments** (Ctrl+Shift+L)— It is possible to set comments at semi-precise locations while playing your scenario in-game. After doing so, you can import them into your scenario using the Import Comments command. For example, a designer is walking through their level and they see a single tree placed upside down. They make a comment, then import comments in Sapien and they can easily find the location of the offending tree.
 - **Sync Mission**(Ctrl+I)— Gets the latest version of the current scenario from source depot.
 - **Submit Mission** (Ctrl+U)— Checks a scenario back into the depot (if you had it checked out for editing).

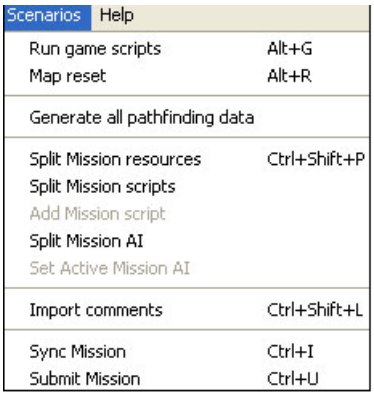
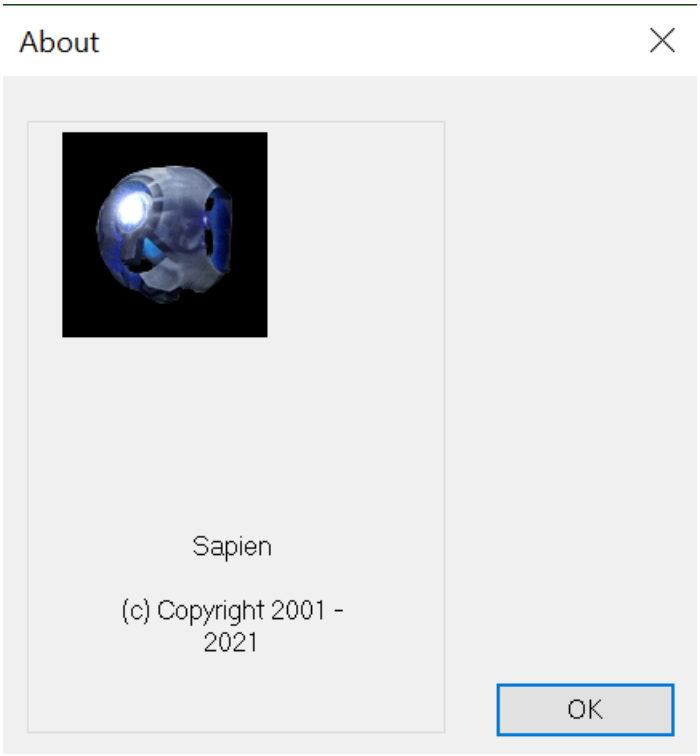


Figure 5 - Sapien's Scenario Menu

- **Help**—



Game Window

11/23/2022 • 2 minutes to read

The Game Window in Sapien (see Figure 1) is where you can see all of the changes you make to your scenario. It provides a visual way for you to place and position assets, as well as preview scripts and AI interactions. There are pointers (the location of the mouse pointer) and camera coordinates in the status bar (along the bottom) of the window.

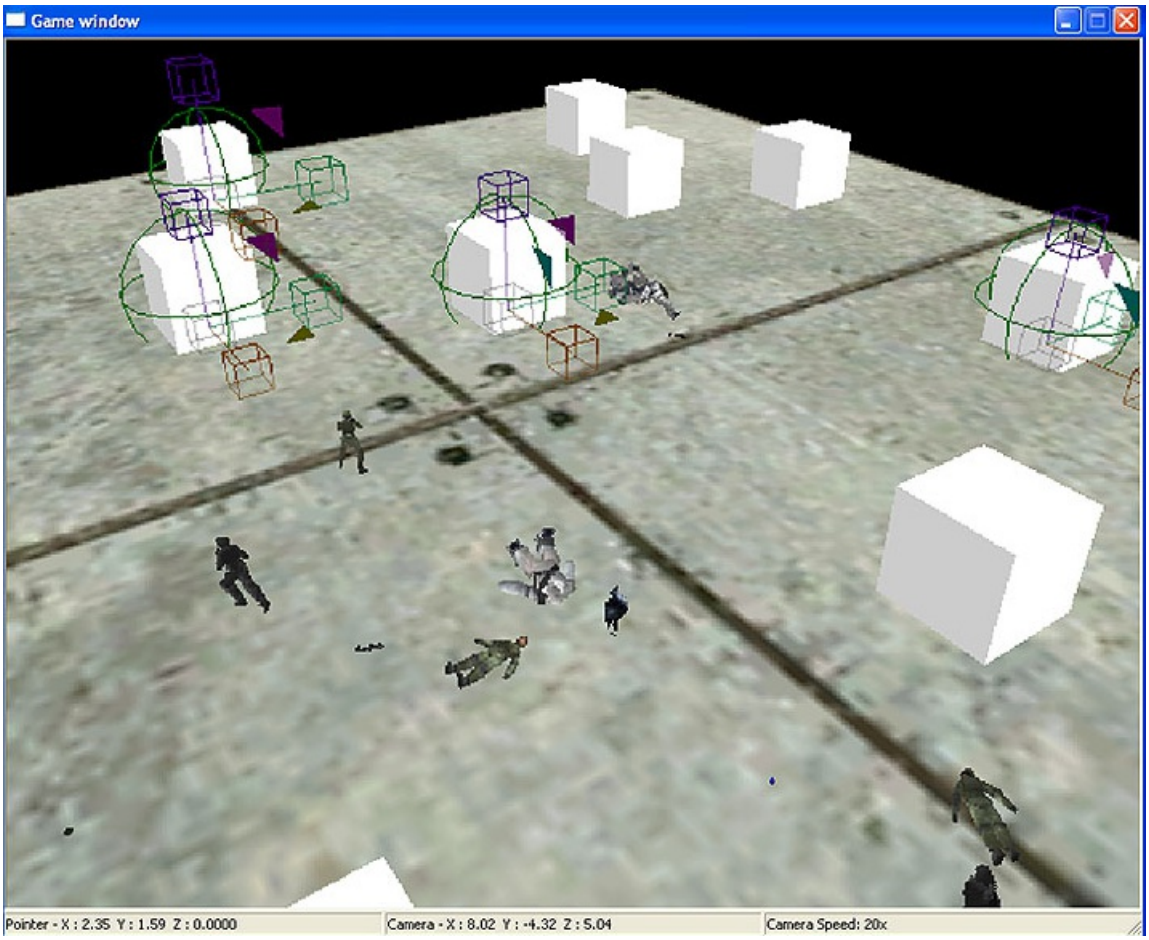


Figure 1 - The Game Window in Sapien

Controls

Here's how to get around within the game window:

- **Turn/Rotate Camera**— Middle Mouse Button + Mouse in direction to turn
- **Move Camera Left/Right/Forward/Back**— Middle Mouse Button + W,A,S,D keys
- **Move Camera Up**— Middle Mouse Button + R key
- **Move Camera Down**— Middle Mouse Button + F key
- **Change Camera Speed**— Middle Mouse Button + Shift (or alternately, you can scroll the mouse wheel)
- **Debug Menu**— Middle Mouse Button + Home
- **Quick Zoom**— Double-click left mouse button

Most of the interaction available via the Game Window is very contextual. In general, left-clicking on an asset will select it (once that type is selected in the hierarchy view) and right-clicking in the game window will place an asset of the type that is currently selected in the hierarchy view. For more information about positioning and re-

sizing assets, see the *Sapien Asset Manipulation Gizmo* article.

Hierarchy View

11/23/2022 • 2 minutes to read

Sapien's Hierarchy view (see Figure 1) is the main interface for placing and selecting assets in your scenario. It looks similar to a windows explorer window with an asset type tree in the left pane and individual asset information in the right pane. Keep in mind that the folders are not actually file folders which reside on your hard drive— they are merely visual representations of the assets which are contained within your scenario (either within your .scenario tag, or within the resource tags if you've split your mission resources). Also contained in the hierarchy view are three buttons: Edit types, New Instance, and Delete.

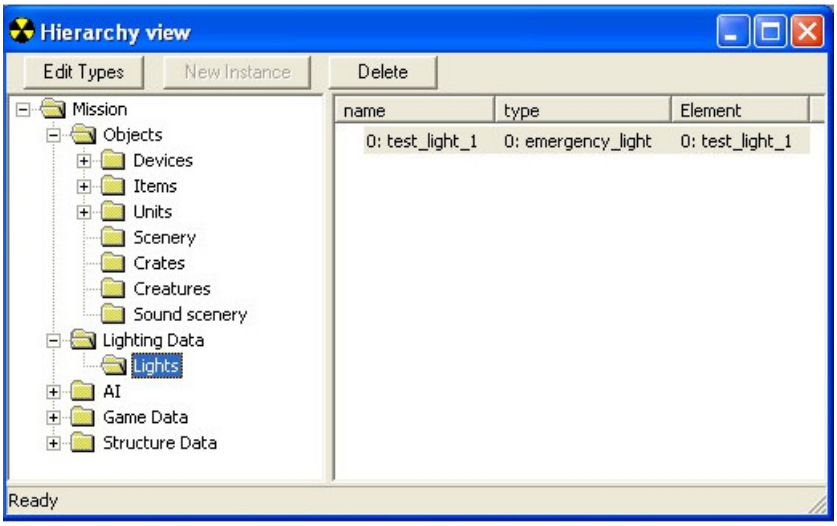


Figure 1 - The Hierarchy View window in Sapien

The Asset Type Palette

Sapien (and the Halo engine) only loads those game assets for each scenario that it needs, so in order to place a specific asset, you first need to add it to the set of assets that Sapien loads for your scenario. That's where the Asset Type Palette (see Figure 2) comes in. Basically, it's the list that Sapien looks at to know which assets to load into memory when you launch your scenario. So, if you want to use something in your scenario, you have to add it to the palette first.

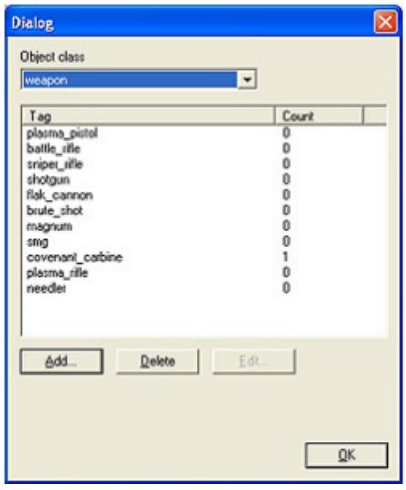


Figure 1 - The Asset Type Palette

To access the Asset Type Palette (and add an asset type to your scenario), do the following:

1. In the Hierarchy View window, click the **Edit Types** button.
2. A dialog similar to Figure 2 should appear. Select the category of object you want to add to your scenario from the Object Class drop down list.
3. Click the **Add** button.
4. Browse to the location of the correct tag and click the **Add Tags** button. When you're finished adding tags, click **Done**.
5. Back in the Asset Type Palette, you should see the name of the asset you just added to your scenario. When you're finished adding assets, click **OK**. You won't have any sort of visual representation of the asset you added to your scenario in the Hierarchy View, but you can always go back to the Asset Type Palette and use the drop-down list to check which assets you've added for each category.

> **IMPORTANT**
> Once you've added an asset to the palette, it will always be loaded when you launch your scenario— even if you never actually place an instance of that asset!

Assets and Instances

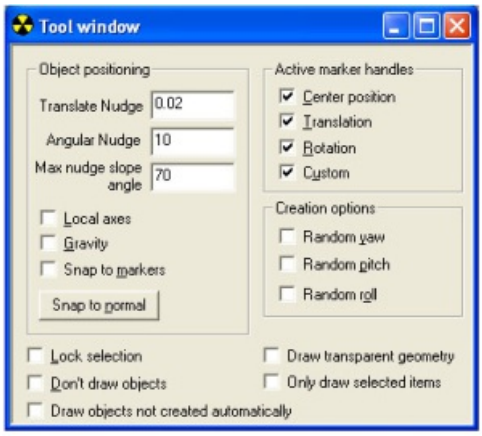
Once you have added assets to the palette for your scenario, you need to place instances of those assets where you want them. There are a couple of different ways to place instances. For objects, you can place an instance simply by selecting the category in the Hierarchy View window (say, objects\scenery for example) and then right-clicking in the Game Window. This will place an instance of a scenery object. Of course, after placing it you'll have to use the Properties Palette to set the type (and only asset types you've added to your palette will be available). For Assets that aren't objects — such as AI squads — you need to select the category in the Hierarchy View window (ai\squads, for example) and then click the **New Instance** button.

For more information on placing objects, see the *Placing Objects* in Sapien article.

Tool Window

11/23/2022 • 2 minutes to read

The Tool Window shows the properties for Sapien's *Asset Manipulation Gizmo*. It allows you to change the way the gizmo functions in selecting and positioning various assets within your scenario.



Controls

Object Positioning

- **Translate Nudge**— Currently not supported, does nothing.
- **Angular Nudge**— Currently not supported, does nothing.
- **Max Nudge Slope Angle**— Currently not supported, does nothing.
- **Local Axes**— Sets the axes on the manipulate gizmo to match the rotation of assets. If this is not checked, the Manipulate gizmo axes are positioned universally and don't move with changes to an asset's position.
- **Gravity**— Disables the translation (move) handles on the manipulate gizmo.
- **Snap to Markers**— Currently not supported, does nothing.
- **Snap to Normal**— Snaps the Pitch (P) and Roll (R) axes to 0.

Active Marker Handles

- **Center Position**— Enables/Disables use of the grey box center handle (free move) of the manipulate gizmo.
- **Translation**— Enables/Disables use of the translation/move handles (Green, orange, and purple boxes) of the manipulate gizmo.
- **Rotation**— Enables/Disables use of the rotation handles (yellow, magenta, and turquoise triangles) of the manipulate gizmo.
- **Custom**— Only used when setting trigger volumes. Checking this flag allows you to resize particular faces of trigger volumes (otherwise you can only scale the whole volume).

Creation Options

- **Random Yaw**— When this box is checked, any asset placed in the scenario (via right-click) will be placed with a randomly generated yaw setting.
- **Random Pitch**— When this box is checked, any asset placed in the scenario (via right-click) will be placed with a randomly generated pitch setting.
- **Random Roll**— When this box is checked, any asset placed in the scenario (via right-click) will be placed with a randomly generated roll setting.

View Options

- **Lock Selection**— Locks the currently selected asset so that you can't accidentally select another asset.
- **Don't Draw Objects**— Disables drawing objects in Sapien.
- **Draw Objects Not Created Automatically**— For objects, it is possible to set a "Not automatically" flag in the Properties Palette. This command causes those objects to render in the Game Window. This is a legacy command that was due to Sapien not being able to handle rendering all objects (not necessary now).
- **Draw Transparent Geometry**— Renders transparent geometry— such as glass— in the Game Window. Not a necessary setting anymore because transparent geometry is automatically rendered (although it is currently rendered opaque).
- **Only Draw Selected Items**— Only Renders (draws) the currently selected asset in the Game Window.

Asset Manipulation Gizmo

11/23/2022 • 2 minutes to read

The Asset Manipulation gizmo in Sapien accomplishes a variety of tasks: It can move, rotate, resize, and reshape various elements within your scenario. The operation of the Asset Manipulation Gizmo is closely tied to the *Tool Window*. Refer to that article for more information on changing properties of the manipulate gizmo.

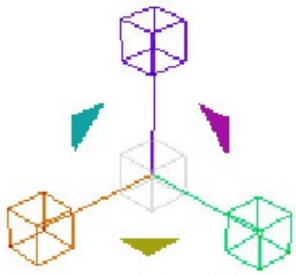


Figure 1 - Sapien's Asset Manipulation Gizmo

Use

Rotate— You can rotate an asset with the manipulate gizmo along 3 different axes, represented by the three colored triangles (yellow, magenta, and turquoise). To rotate an object, mouse over it until it enlarges, then click and drag to your desired rotation. The rotation function can also be slightly contextual. For example, when placing a light object, you'll need to press the SHIFT key to access the rotate handles.

-  — The yellow triangle rotates the asset around its **Y** axis (Y = Yaw). This results in sort of spinning the asset.
-  — The magenta triangle rotates the asset along its **R** axis (R = Roll). It tilts the object from side to side (or front-back depending on the direction you're looking at it from).
-  — The turquoise triangle rotates the asset along its **P** axis (P = Pitch), resulting in a tilt from forward-back (or side-to-side depending on the direction you're looking at it from).

Move (also known as Translation)— You can move assets along their X, Y, and Z axes by mousing over one of the gizmo's four cubes until it turns a semi-solid color, then click and drag to move the asset along that particular axis.

-  — Click the green cube and drag to move the asset along the Y axis.
-  — Click the orange cube and drag to move the asset along the X axis.
-  — Click the purple cube and drag to move the asset along the Z axis.
-  — Click the grey cube in the center of the gizmo and drag to enter free move mode. This allows you to move the asset along all axes at once and simply place it wherever you want in your scenario.

Scale (Resize)— The Scale/resize function of the manipulation gizmo is accessed by pressing the ALT key (and in some cases the CTRL key) and changes based on the type of asset selected. For objects (such as a simple box), the scale tool creates a sphere that matches the bounding sphere of the object (See Figure 2). When moused over, the sphere turns red. Then, you can click and drag to scale the object larger or smaller. For other objects, such as a custom light, there can be more than one scale implementation and you need to use either ctrl or alt to access them— this is so you can resize various planes of the light. See Figure 3 for an example of the scale

function on a custom light object.

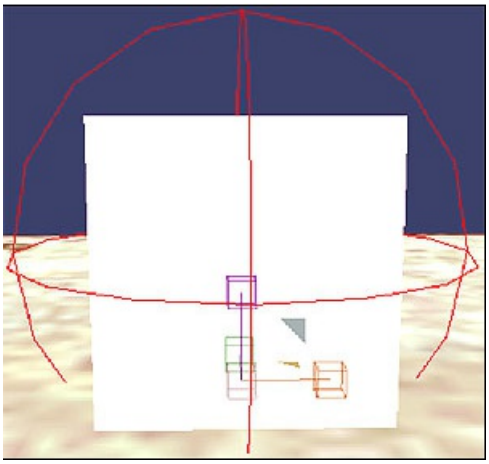


Figure 2 - The scale function of the gizmo with a box scenery object selected

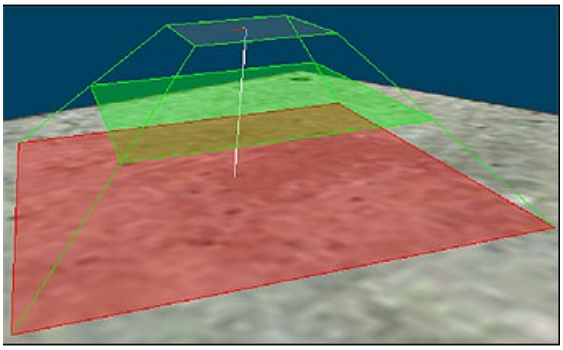


Figure 3 - The scale function of the gizmo on a custom light object

On a final note, while the asset manipulation gizmo provides a visual reference for making changes to assets, it is also important to note that the changes can be made with more precision via the position, rotation, and scale attributes in the *Properties Palette* window.

Output Window

11/23/2022 • 2 minutes to read

The Output Window in Sapien spills out script compiling debug information. Specifically, if there are any errors when compiling your scripts, it provides information about the problem. See Figure 1 for an example.

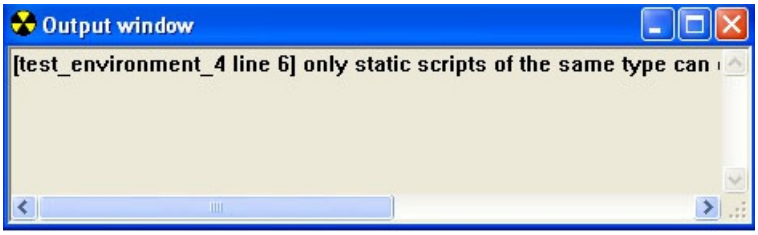


Figure 1 - The Output Window in Sapien

Placing Objects

11/23/2022 • 2 minutes to read

To place an object in a scenario using Sapien, do the following:

1. Launch your scenario in Sapien.
2. In the Hierarchy View window, expand the Mission folder, then the Objects folder. Click on the category of object (Scenery, Crate, Creature, Device, etc.) you want to add to highlight it.
3. In the Hierarchy View window, click the **Edit Types** button.
4. From the drop-down list in the *Asset Type Palette*, select the type of object you want to add then click the **Add** button
5. Find the tag for the appropriate object and click **Add Tags**. Click **Done** and then **OK** to close the asset type palette.
6. Now that you've added the object to your scenario, it's time to place an instance of it. In the Hierarchy View window, click the category of object you just added to highlight it.
7. In the Game Window, right-click to place an object. You'll notice that when the object is placed, an entry appears in the hierarchy view (with Type: None), but in the Game Window only the manipulate gizmo appears. This is because you need to tell Sapien what the type of the object is you just placed.
8. With your object still selected (you can click on its entry in the Hierarchy View Window to be sure), click on the **Type** drop down menu (Figure 1) in the **Properties Palette** window. Select the object you added to your palette. If your object doesn't appear, you most likely added an object from the wrong category in the Hierarchy View.

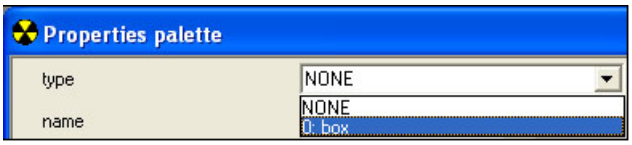


Figure 1 - The Type drop down menu in the Properties Palette window

9. Once you've selected a type, you should see your object appear in the Game Window. For information on how to move, scale, or rotate your object, see the *Asset Manipulation Gizmo* article.
10. Save your scenario.
11. If you're connected to an Xbox, you should Xsync (shift-ctrl-s) and look at the object you just placed in your scenario in-game.

Placing Lights

11/23/2022 • 2 minutes to read

To place a light in a scenario using Sapien, do the following:

1. Launch your scenario in Sapien
2. In the Hierarchy View window, expand the Mission folder, then the Lighting Data folder. Click on the Lights folder to highlight it.
3. In the Hierarchy View window, click the Edit Types button.
4. From the drop-down list in the Asset Type Palette, select the type of object you want to add. Click the Add button
5. Find the tag for the light you want and click Add Tags. Click Done and then OK to close the asset type palette.
6. Now that you've added the light to your scenario, it's time to place an instance of it. In the Hierarchy View window, click the category of object you just added to highlight it (if it isn't already selected).
7. In the Game Window, right-click to place a light. A generic light object (see Figure 1) will appear in the Game Window. In the Hierarchy View, you'll notice that a new light object has been added to the list, but under type it says none. You still need to set the type of the light in the Properties Palette window.

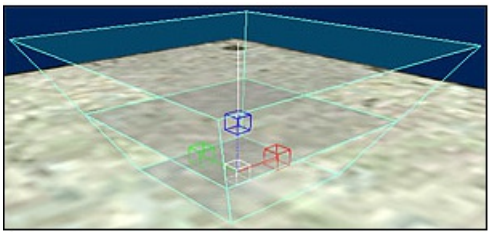


Figure 1 - A Generic light object placed in the scenario.

8. With your light still selected (you can click on its entry in the Hierarchy View Window to be sure), click the Type drop down menu (see Figure 2) in the Properties Palette window. Select the type of light you added to your palette.

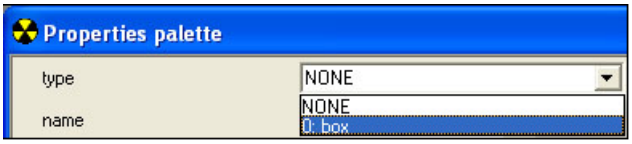


Figure 2 - The Type drop down menu in the Properties Palette window.

9. Once you've selected a type, you should see your particular light appear in the Game Window (although some lights simply remain the same shape as the generic one). For information on how to move, scale, or rotate your object, see the Asset Manipulation Gizmo article.
10. Save your scenario.
11. If you're connected to an Xbox, you should Xsync (shift-ctrl-s) and look at the object you just placed in your scenario in-game.

Resizing and Positioning a Light

Most aspects of a light object can be controlled using the *Asset Manipulation Gizmo*, including the size, shape, and brightness planes. However, in order to adjust all these properties, the light first needs to be set as **Custom Geometry** — otherwise you'll be limited to simply rotating and moving it. You can set a light as custom geometry by clicking the **Custom Geometry** checkbox in the *Flags* section of the Properties Palette window

(see Figure 3).



Figure 3 - The Custom Geometry Flag

After your light is set as custom geometry, its blue lines turn green and it has two Manipulate gizmos (instead of just one). Now you can hold down the alt key to adjust the height of the light planes, and hold down the ctrl key to adjust the size of the light cone. Press shift to access the Manipulate gizmo's rotate handles. Also, remember that you can adjust any of these properties in the *Properties Palette* window with more precision.

Properties Palette

11/23/2022 • 2 minutes to read

The Properties Palette window in Sapien (see Figure 1) allows you to edit a myriad of properties for every asset you place within your scenario. It is completely contextual, so the properties change based on the asset you have selected in the Hierarchy View. The properties palette shown in Figure 1 is an example of what the palette looks like with a simple box object selected. See individual articles for information on which properties need to be configured for your asset to work correctly.

You may notice that some blocks of the properties palette are greyed out. This is for your safety and protection, but if you must edit one of those properties (and you know what you're doing), you can access them by turning on **Expert Mode** in the Edit menu.

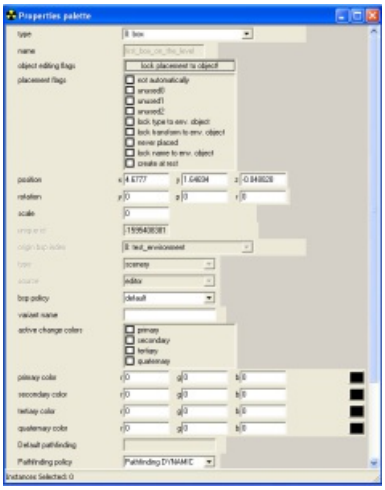


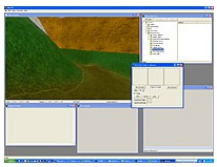
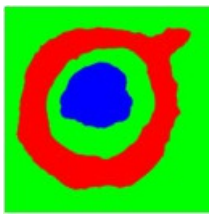
Figure 1 - The Properties Palette Window

Structure Painter

11/23/2022 • 4 minutes to read

The Structure Painter tool is used to dynamically create or edit a blend map for your scenario while in Sapien. It gives you the ability to paint bitmaps where you want them in the scenario (while creating the blend map for you).

How it Works

SCENARIO IS OPENED IN SAPIEN.	--	PAINT IN THE GAME WINDOW	--	STRUCTURE PAINTER WRITES TO A .TIF FILE (BLEND MAP)
				

- The structure painter loads up the .tif files referenced by the terrain shader attached to your mesh. The structure painter writes in the red, green, blue, and alpha channels of the .tif file— just as if you were painting it manually in Photoshop.
- Anything you paint in the game window using the structure painter gets written out to a blend_map .tif file. The Structure painter has no effect on your shader tags, or the .tif files referenced by them (unless you choose to save over the top of a blend map you were previously using).
- The structure painter doesn't save or change anything automatically! You have to create a blend_map .tif file and link that to a .shader_terrain before you'll be able to paint with the structure painter.
- The structure painter saves to a .tif file. It will need to be imported as a .bitmap tag manually.

User Interface

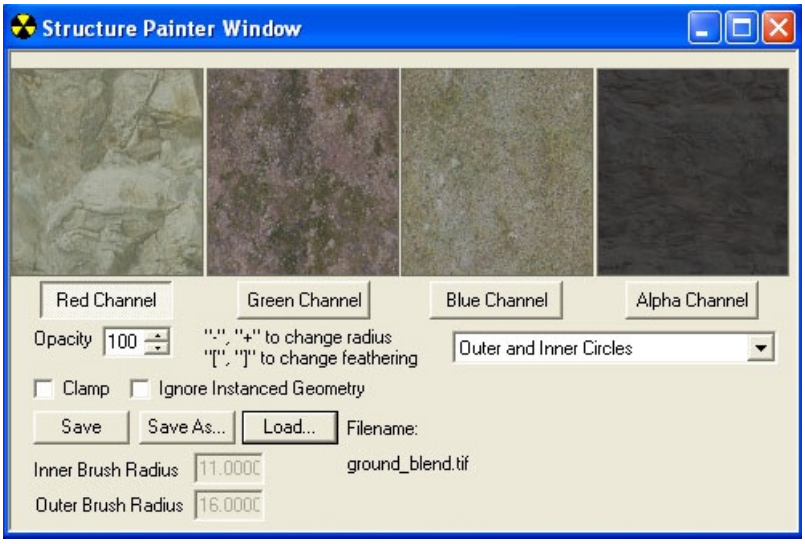


Figure 2 - The Structure Painter Tool inside Sapien

- **Image Previews**— The image previews show the bitmap that will be painted in the game window when the corresponding channel button is selected. The bitmap files are loaded from those in the terrain shader in

Guerilla. If you have more than one terrain shader in your scenario, the image previews will change when you move the mouse over the different shaders in Sapien.

- **Channels**— When clicked, the channel button selects the color channel to paint (Red, Green, Blue, or Alpha) in the .tif file. It will paint the image loaded into the image preview (above the button) in the game window.
- **Opacity**— The Opacity spinner (and text box) lets you set the opacity of the brush for the currently selected channel.
- **Clamp**— Unchecking the clamp checkbox allows you to paint over the top of an area without replacing what is already there. This would be akin to painting a value in more than one channel of the blend map — thus over-saturating it if you don't use this in conjunction with lowering the opacity. If you're not careful, it's an easy way to end up with a big black mess.
- **Ignore Instanced Geometry**— When this is checked, the structure painter will ignore any instanced geometry.
- **Radius**— The size of the brush. In the inner circle, the brush will paint with 100% opacity (unless overridden by the opacity setting). Use the + and - keys on the keyboard to change the size of the Radius.
- **Feathering**— The feather radius (the outside circle) is a gradient from 100% value (of whatever the inner radius is) to 0%. The size of the feather radius can be adjusted using the [and] (bracket) keys.
- **Save**— Saves the current blend map.
- **Save As**— Presents the user with a dialog which allows them to choose the name and location of the blend map .tif file and then save it.
- **Load**— The load button presents you with a dialog that allows you to load a previously created blend map for editing.
- **Brush Display Drop-down**— Allows you to change the way the brush is displayed in the game window. You can choose Preview (shows no rings), Outer Circle (only shows outer edge), or Outer and Inner Circles (shows both the brush radius and the feathering radius).

Step-By-Step

Here's how to create a blend map using the Structure Painter Tool:

1. Create a new blend map .tif file in Photoshop (it needs at least a small bit of information in it or it won't import correctly). Name it with _blend on the end and then import the bitmap.
2. Create a new .shader_terrain shader and set the blend map you just created and imported as the blend_map.
3. Assign bitmaps to the material slots in your shader_terrain as necessary.
4. Launch your scenario in Sapien.
5. In the Hierarchy View, expand **Scenario -> Structure Data**, then click on **Structure Painter**. This will open the Structure Painter window. If the bitmaps you want to use are not loaded automatically into the image preview slots, you need to change them in the shader in guerilla.
6. Once you've got the images loaded, click on one of the Channel (red, green, blue, or alpha) buttons to select it.
7. Go to the game window and begin painting. If you want to increase or decrease either brush radius size, use the [or] key (or shift + [or] for the outer brush radius).
8. When you're finished painting, click the **Save** button.
9. Switch to **Guerilla** and **import** the blend map .tif file to refresh your .bitmap tag. If you skip this step, you will paint over the top of your blend_map without realizing it! Always remember: the Structure Painter is painting the .tif file and gives you a preview on the screen of what the imported .bitmap will look like - it doesn't make changes until you save and re-import.
10. Xsync and check out your new detail blend!

AI

11/23/2022 • 2 minutes to read

This document serves to provide an overview of what to expect from the AI in Halo 3. Various other documents linked throughout this page will give more details on specific areas of the AI.

One of the main aims of the AI in Halo 3 is to provide players with familiar encounters, but to also provide new variations on the standard encounters. In Halo and Halo 2 an encounter would involve entering a space, confronting some low level characters, killing some of them, watching them fall back, pressing the enemies, killing more, and eventually mopping everything up. In Halo 3, we introduced ways to allow designers to vary how an encounter plays by making very small changes.

AI Activities

A list of the activities AI can perform while not in combat.

AI Brute Variations

Breakdown of the Brute variations introduced in Halo 3.

AI Character Variant Names

Lists of all of the variant names for each character in Halo 3

AI Combat Status

AI Combat Status Values

AI Debug Commands

AI debugging commands.

AI Difficulty Levels

Common game variables modified by difficulty.

AI Exhaustion and Max Duration

Descriptions of the Exhaustions and Max Duration features.

AI Leadership

Information on the leadership system for AI squads.

AI Objective Window Color Key

Explanation of what all of the various colors in the Objective window within Sapien mean.

AI Relaxation Poses

Frame and Pose information for setting up .jma files.

AI Task Status

A list of states for the ai_task_status command.

Activities

11/23/2022 • 2 minutes to read

Below, you'll find a list of the activities AI can perform while not in combat. Each activity includes a description, screenshots of each character performing it, and a list of which characters can perform it.

Patrol

- **Description** — Character walks between pre-designated patrolling points.
- **Who can use it** — Brute, Bugger, Elite, Floodcombat_elite, Floodcombat_human, Grunt, Hunter, Jackal, Marine, Monitor, Sentinel_aggressor.





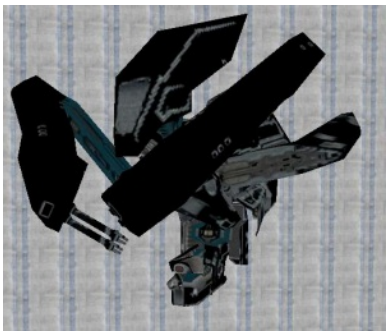












Sleep

- **Description** — Character sleeps.
- **Who can use it** — Grunt.



Kneel

- **Description** — Character kneels.
- **Who can use it** — Brute, Bugger, Elite, Grunt, Jackal, Marine.








Guard

- **Description** — Character stands guard.

- **Who can use it** — Brute.



Corner

- **Description** — Character corners.
- **Who can use it** — Brute, Elite.





Injured

- **Description** — Character acts badly wounded.
- **Who can use it** — Marine.



At Ease

- **Description** — Character is at ease.
- **Who can use it** — Marine.



Cower

- **Description** — Character cowers in fear.
- **Who can use it** — Marine.



Tai Chi

- **Description** — Character performs a type of covenant martial art which looks similar to Tai Chi.
- **Who can use it** — Brute, Grunt, Jackal.

Brute Variations

11/23/2022 • 2 minutes to read

Ranger (brute_jumppack)

The Brute Ranger's distinguishing factor is an enormous jetpack strapped onto his back. This jetpack isn't used for flying, but instead allows the Ranger to perform boosted jumps, significantly extending how high and far he can jump. The jetpack is used for a number of key behaviors.

- **Height** — The Ranger is designed for long range combat; ideally he will be equipped with a sniper rifle or similar and will be given a large outdoor area to snipe in. He will use his jetpack to jump to a high location, take a couple of shots at his target, and then move to another position.
- **Avoidance** — Instead of diving to the side when confronted with an immediate danger such as a grenade or a vehicle running straight at him, the Ranger will make use of his jetpack to jump to safety, either straight up or to one side.
- **Insertion** — The Ranger has the ability to use his boosted jump to cover large distances very quickly. This will result in him uncovering a target by performing a boosted jump and landing very close to his target, coming in from an unexpected vector. Designers should also try and make use of this ability to provide impressive appearances of Rangers.
- **Charging** — When berserking the Ranger will use his jetpack to jump right at the target, ending in a melee swipe that should cause considerable damage.

Because of the Ranger's ability to travel at high speeds and to otherwise inaccessible areas, he may well be the first character in an encounter that the player sees, and will likely provide information to the rest of the pack about the target's location. Taking out the Ranger early means that the player no longer has to worry about attacks from unexpected directions.

Tracker (brute_stalker)

The Brute Tracker has an uncanny sense of smell, and has the ability to perform long distance, relentless searches for his target. When guarding or otherwise idle, he will be seen to periodically sniff the air or check the ground for tracks. Should he detect an enemy within a certain range, visible or not, he will alert the pack and launch a search for the target. While the Tracker is searching no other characters in that group will be searching, instead leaving the Tracker and his heightened detection skills to hunt down the target.

While searching the Tracker will sometimes pause to redetect the scent of his target, again indicated by some animation. He will relay this information to the rest of the pack, and then continue with his search. When a Tracker is present, it should be virtually impossible flank an encounter, the pack should be waiting for the target wherever they emerge from cover.

Killing the Tracker early in an encounter will be key to a player's survival. With a Tracker on their heels they will be unable to seek cover and recharge their shield for a significant amount of time, and they will be unable to surprise a group of characters and attack from the rear or a flank.

Character Variant Names

11/23/2022 • 2 minutes to read

Below are lists of all of the variant names for each character in Halo 3:

objects\characters\brute\brute.model

- minor
- captain
- jumppack
- stalker
- chieftain_armor
- chieftain_weapon
- tracker
- major
- ultra
- captain_ultra
- bodyguard
- captain_major

objects\characters\bugger\bugger.model

- minor

objects\characters\cortana\cortana.model

- default

objects\characters\dervish\dervish.model

- default_arb

objects\characters\elite\elite.model

- minor_scl
- minor_dog
- major_scl
- major_dog
- spec_ops_scl
- spec_ops_dog

- stealth_scl
- stealth_dog
- stealth_major_scl
- stealth_major_dog
- soc_cinematic
- soc

objects\characters\flood_infection\flood_infection.model

- default

objects\characters\flood_ranged\flood_ranged.model

- n/a

objects\characters\flood_stalker\flood_stalker.model

- n/a

objects\characters\flood_tank\flood_tank.model

- default

objects\characters\floodcarrier\floodcarrier.model

- default
- shielded

objects\characters\floodcombat_elite\floodcombat_elite.model

- standard_fce

objects\characters\floodcombat_human\floodcombat_human.model

- standard_fch

objects\characters\grunt\grunt.model

- minor
- major
- ultra
- spec_ops
- heavy

objects\characters\hunter\hunter.model

- minor

objects\characters\jackal\jackal.model

- minor
- sniper
- shieldless
- major

objects\characters\lord_hood\lord_hood.model

- default
- no_hat
- dress
- dress_no_hat

objects\characters\marine\marine.model

- johnson
- wounded
- pilot
- sarge_lehto
- johnson_combat
- johnson_dress
- default
- ma4
- ma6
- blind
- ma2
- sg1
- ma6
- ma3
- ma1
- dead
- sg2
- cinem1
- cinem2
- cinem3
- cinem4

- cinem5
- cinem6
- cinem7
- cinem8a
- cinem8b
- cinem8c
- cinem8d
- cinem8e
- cinem8f
- cinem8g
- cinem8h
- cinem8i
- cinem8j
- johnson_beatup
- johnson_dead

objects\characters\marine\marine_female\marine_female.model

- fem1
- fem3
- fem2
- no_helmet
- anete
- chan
- hartmann
- michelle
- cinef1
- cinef2
- default

objects\characters\masterchief\masterchief.model

- default
- cinematic

objects\characters\miranda\miranda.model

- default

- combat
- dress
- dead

objects\characters\monitor\monitor.model

- monitor
- monitor_combat

objects\characters\truth\truth.model

- default
- floodified

objects\characters\sentinel_aggressor\sentinel_aggressor.model

- default

objects\characters\sentinel_constructor\sentinel_constructor.model

- default

Combat Status

11/23/2022 • 2 minutes to read

AI Combat Status Values:

- _asleep = 0
- _idle = 1
- _alert = 2
- _active = 3
- _uninspected = 4
- _definite = 5
- _certain = 6
- _visible = 7
- _clear_los = 8
- _dangerous = 9

Debug Commands

11/23/2022 • 2 minutes to read

Below are some AI debugging commands that you may find useful:

- Ai_render_all_actors 1
- Ai_render_objectives 1
- Ai_render_paths 1
- Ai_render_command_scripts 1
- Ai_render_sector_bsps 1

Difficulty Levels

11/23/2022 • 2 minutes to read

AI Parameters Modified by Difficulty

There are a number of attributes in the game varied by difficulty. Most of these are related to AI, and happily most of them are also accessible in various tags. Here follows a list of the most common game variables modified by difficulty.

Character (*.character tag)

Vitality

AI characters may have varying levels of health and/or shield based on difficulty level.

Accuracy

The most important thing about AI accuracy is how the firing parameters blend from inaccurate to accurate at different rates depending on difficulty level.

Upgrade chance

Actors can upgrade to different variants, such as "major." The major characters have a different appearance (difference armor, mostly), and may have different behavior properties (more likely to charge, etc).

Scenario (*.scenario tag)

Upgrade chance

As above, but the default value can be overridden on a per squad basis.

Squad placement numbers

Different numbers or placement of actors can happen depending on the difficulty level.

Task flags

Tasks can be enabled and disabled depending on the difficulty level, meaning squads will occupy different areas, be more aggressive, timid, etc.

Projectiles (*.projectile tag)

AI projectile velocity

Velocity of projectiles fired by AI can be modified based on difficulty level.

Globals (globals.globals tag)

There are a load of global modifiers, some of which aren't actually linked up in code, but here is an exhaustive list. There exist different (or sometimes the same) values for each of the following attributes, for each difficulty level.

Health

- *enemy damage*— Enemy damage multiplier
- *enemy vitality*— Enemy maximum body vitality scale
- *enemy shield*— Enemy maximum shield vitality scale

- *enemy recharge*— Enemy shield recharge scale
- *friend damage*— Friend damage multiplier
- *friend vitality*— Friend maximum body vitality scale
- *friend shield*— Friend maximum shield vitality scale
- *friend recharge*— Friend shield recharge scale
- *infection forms*— Toughness of infection forms (may be negative)

Ranged Fire

A lot of these values are the same across difficulty levels because the firing parameters moved from here into the character tag, as discussed above.

- *rate of fire*— Enemy rate of fire scale
- *projectile error*— Enemy projectile error scale, as a fraction of their base firing error
- *burst error*— Enemy burst error scale, reduces intra-burst shot distance
- *new target delay*— Enemy new-target delay scale factor
- *burst separation*— Delay time between bursts scale factor for enemies
- *target tracking*— Additional target tracking fraction for enemies
- *target leading*— Additional target leading fraction for enemies
- *overcharge chance*— Overcharge chance scale factor for enemies
- *special fire delay*— Delay between special-fire shots (overcharge, banshee bombs) scale factor for enemies
- *guidance vs. player*— Guidance velocity scale factor for all projectiles targeted on a player
- *melee delay base*— Delay period added to all melee attacks, even when berserk
- *melee delay scale*— Multiplier for all existing non-berserk melee delay times

Grenades

grenade chance scale— Scale factor affecting the decisions to throw a grenade grenade timer scale— Scale factor affecting the delay period between grenades thrown from the same encounter (lower is more often)

Placement

major upgrade (normal)— Fraction of actors upgraded to their major variant major upgrade (few)— Fraction of actors upgraded to their major variant when mix = normal major upgrade (many)— Fraction of actors upgraded to their major variant when mix = many

Vehicles

player vehicle ram chance— Chance of deciding to ram the player in a vehicle

Exhaustion and Max Duration

11/23/2022 • 2 minutes to read

Exhaustion Delay:

This feature instructs a unit to not enter a new task for the time specified.

Max Duration:

This feature instructs a unit to stay in its task for no longer than the time specified.

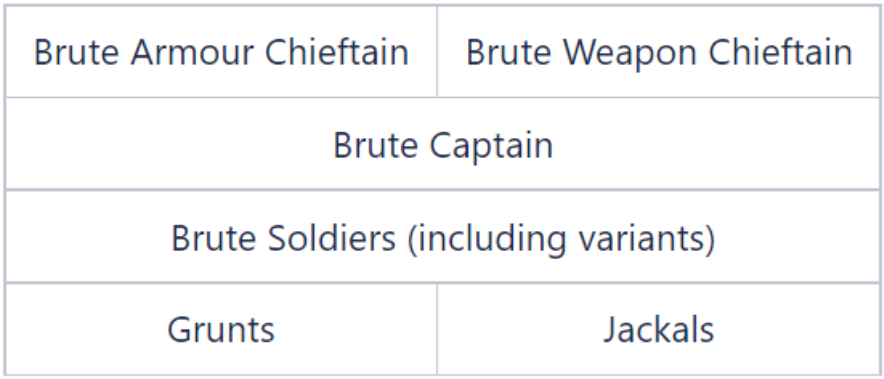
Leadership

11/23/2022 • 4 minutes to read

The aim of the leadership system is to show the player a more rigid command hierarchy within the AI characters. The majority of this will be manifested in Covenant troops, but it is applicable for all characters, for example the marines. The leadership system looks to formalise the combat cycle somewhat, and provide variation from standard encounters with no leadership present. Additionally it introduces a new set of behaviours which indicate that an encounter has been broken, again increasing the variety of encounter mechanics without requiring increased variations in character types.

Who's in charge?

When a group of characters shares an objective, the highest ranked character in that group will be assigned the leadership of all the other characters in that objective. This assignment is decided based on rank levels specified by designers, and typically the hierarchy looks something like this.



If any of the higher levels are present, they will take command. If more than one of the highest level is present, one will be chosen as the leader and the other will be considered "peers". Everyone else will be considered "followers".

The following table lists the leadership rank and scariness value assigned to all of the .character tags.

Rank and Scariness

CHARACTER	LEADERSHIP RANK	SCARINESS
Brute	50	4
Brute Chieftain	100	6
Brute Jumppack	NA	NA
Brute Major	75	5
Brute Soldier	0	3
Brute Stalker	NA	NA
Brute Tracker	NA	NA

CHARACTER	LEADERSHIP RANK	SCARINESS
Bugger	0	0
Cortana	0	0
Dervish	NA	NA
Elite	0	4
Elite Councilor	NA	NA
Elite Honor Guard	NA	NA
Elite Major	NA	NA
Elite Ranger	NA	NA
Elite SpecOps	NA	NA
Elite SpecOps Commander	NA	NA
Elite Stealth	NA	NA
Elite Stealth Major	NA	NA
Elite Ultra	NA	NA
Elite Zealot	NA	NA
Flood Infection	0	0
Flood Pureform	0	4
Flood Pureform Ranged	0	4
Flood Pureform Stalker	0	4
Flood Pureform Tank	0	4
Flood Carrier	0	4
Flood Carrier Major	NA	NA
FloodCombat Elite	0	8
FloodCombat Elite Shielded	NA	NA
FloodCombat Elite Weak	0	8
FloodCombat Human	NA	NA

CHARACTER	LEADERSHIP RANK	SCARINESS
FloodCombat Human Weak	NA	NA
Grunt	0	0
Grunt Heavy	NA	NA
Grunt Major	NA	NA
Grunt SpecOps	NA	NA
Grunt Ultra	NA	NA
Guardian	0	16
Hunter	0	16
Hunter Shielded	0	20
Jackal	0	0
Jackal Major	NA	NA
Jackal Sniper	NA	NA
Marine Crewman	0	0
Marine	0	3
Marine Dress	NA	NA
Marine Female	NA	NA
Marine Johnson	100	3
Marine Johnson Boss	NA	NA
Marine Johnson Dress	NA	NA
Marine Massive	NA	NA
Marine ODST	0	4
Marine	50	3
Marine Wounded	NA	NA
Masterchief	0	7
Miranda	0	4

CHARACTER	LEADERSHIP RANK	SCARINESS
Monitor	0	0
Prophet of Truth	0	4
Sentinel Aggressor	0	4
Sentinel Aggressor Eliminator	NA	NA
Sentinel Aggressor MajorNA	NA	
Sentinel Aggressor MP	0	4
Sentinel Constructor	0	0
Sentinel Enforcer	0	16

Leadership Role

When a character is in a leadership role, their basic aim is to stick near the back of the encounter and direct the action until all the followers are dead, at which point the player will engage the leader providing a mini boss battle. Typically the leaders will be Brute Chieftains, who will have special attacks and exciting weapons.

It should be possible to identify which character is the leader not only by their appearance but also by their positioning and the positioning of their followers with respect to them. The leader is given a specific subset of firing positions which he uses for his movement, which should be towards the back of the encounter and near any "objectives" that the group of enemies are protecting, such as the door through which the player must progress. Followers should form up in front of the leader, spreading out in a fan protecting them from attack.

Having a leader also formalizes the combat cycle of the group somewhat. There will be a pause before engaging, waiting for the leader to order his minions to attack. Similarly the leader will order his followers to search for the target, and to uncover the target using a barrage of grenades.

Breaking the Encounter

One of the key features of the leadership behavior is the ability to break a pack of enemies. Should the player succeed in killing the leader prematurely, his followers should let out wails and then break. This "breaking" is indicated in different ways depending on the character type.

- **Brutes**, *Scatter*. Any Brutes who are broken should flee for cover, or if in close proximity to the target they will charge and melee the attacker. After a few seconds they will emerge from cover and continue attacking as normal.
- **Grunts**, *Flee*. Broken Grunts will turn tail and flee for their lives, searching for cover or in the absence of cover they will just continue bumbling around wildly. Once they are no longer threatened they will calm down and continue attacking as normal.
- **Jackals**, *Huddle*. Jackals will form a huddle when broken, in which they will gather round a single central position and crouch down behind their shields firing. Given the concentrated firepower these huddles are potentially very dangerous to a careless target, but a single well placed grenade or a quick meleeing spree will sort everything out. Once they are no longer threatened they will break the huddle and continue attacking as normal.

Squad Patrol

If you have a group of different units who are patrolling as a group, and some of those units are faster than others, this AI feature will have the faster units slow their pace so they don't get too far ahead of the slower units.

Objective Window Color Key

11/23/2022 • 2 minutes to read

Below is an explanation of what all of the various colors in the Objective window (see Figure 1) mean:

- **Red Background:** task is DISABLED
- **Blue Background:** task is a single-use task
- **Grey Background:** task is a Gate Task
- **Red Outline:** task is flagged as Latch Off
- **Green Outline:** task is flagged as Latch On

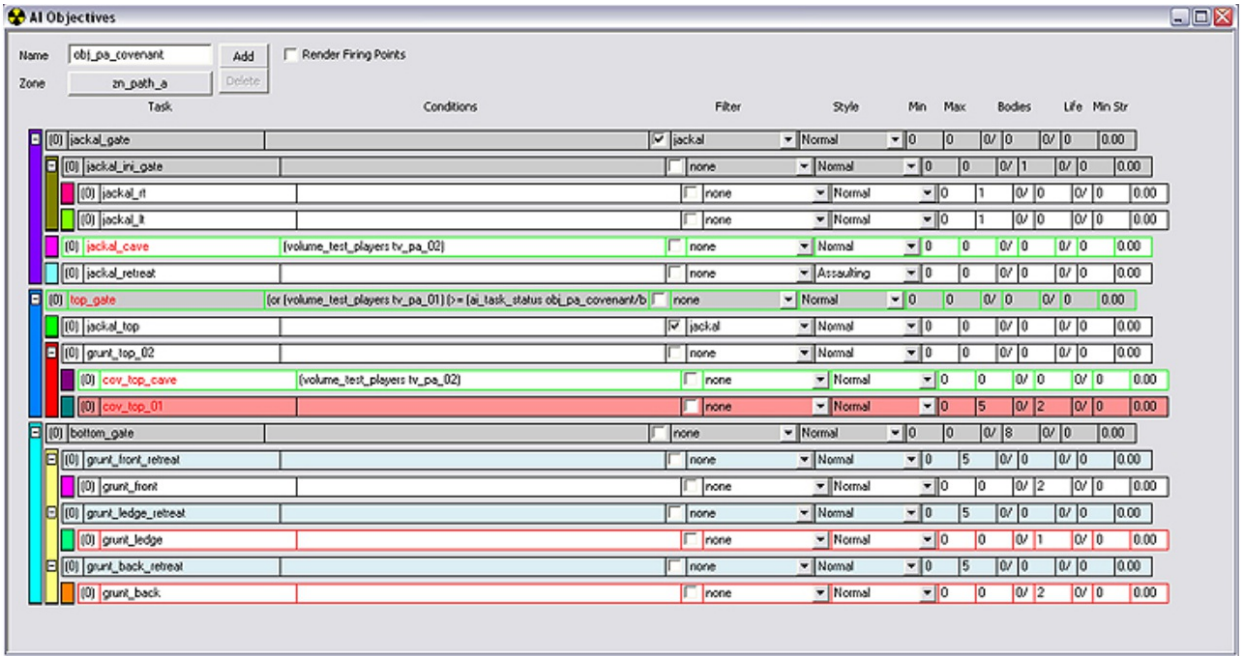


Figure 1 - The Objective Window.

Relaxation Poses

11/23/2022 • 2 minutes to read

Setting up the relaxation_pose.jma file

FRAME	POSE
0	_biped_relaxation_pose_grounded_head,
1	_biped_relaxation_pose_grounded_foot,
2	_biped_relaxation_pose_grounded_front,
3	_biped_relaxation_pose_grounded_back,
4	_biped_relaxation_pose_grounded_left,
5	_biped_relaxation_pose_grounded_right,
6	_biped_relaxation_pose_airborne_head,
7	_biped_relaxation_pose_airborne_foot,
8	_biped_relaxation_pose_airborne_front,
9	_biped_relaxation_pose_airborne_back,
10	_biped_relaxation_pose_airborne_left,
11	_biped_relaxation_pose_airborne_right,

Task Status

11/23/2022 • 2 minutes to read

A list of states for the ai_task_status command.

ai_task_status_

- _never (0)
- _occupied (1)
- _empty (2)
- _inactive (3)
- _exhausted (4)

Art

11/23/2022 • 2 minutes to read

3DS Max

Landing Page for all information related to 3DS Max.

Maya

Landing Page for all information related to Maya.

Animation

Landing Page for all information related to Animations.

Decals

Landing Page for all information related to Decals.

Decorators

Landing Page for all information related to Decorators.

Instance Objects

Landing Page for all information related to Instance Objects.

Lightmaps

Landing Page for all information related to Lightmaps.

Pre-Computed Radiance Transfer

Landing Page for all information related to Pre-Computed Radiance Transfer.

3DS Max

11/23/2022 • 2 minutes to read

Bulk File Import and Export

Bulk File Import and Export

11/23/2022 • 4 minutes to read

Using the bulk file export and import tools is a quick way to export all the pieces of a model from 3ds Max and then import and/or create the tags for them in Guerilla. In addition, the bulk file tools allow you to set up multiple objects within a single 3ds Max file and export (and import) them all at the same time.

The process looks something like this:

Node naming conventions

- !name— Bang Node/World Node
- b_— Frame node
- bip_— Frame node
- frame_— Frame node
- bone_— Frame node
- @name— Collision
- $name— Physics
- name (no symbol)— Render

Models are created in 3ds Max using our standard node structure: A world node (bang node, !), a root (or frame) node (named with a b_), render geometry, collision geometry (specified by using the @ symbol in the name), and physics geometry (specified with a $ in the name). See Figure 1 for an example.



Figure 1 - The correct node structure for an object in 3DS Max

More than one model can be created in the same 3ds Max file using our standard node structure. Each model needs to have its own world node (bang node) for the .jmi file to sort them into the correct .jms files and sub-folders. See Figure 2 for an example.

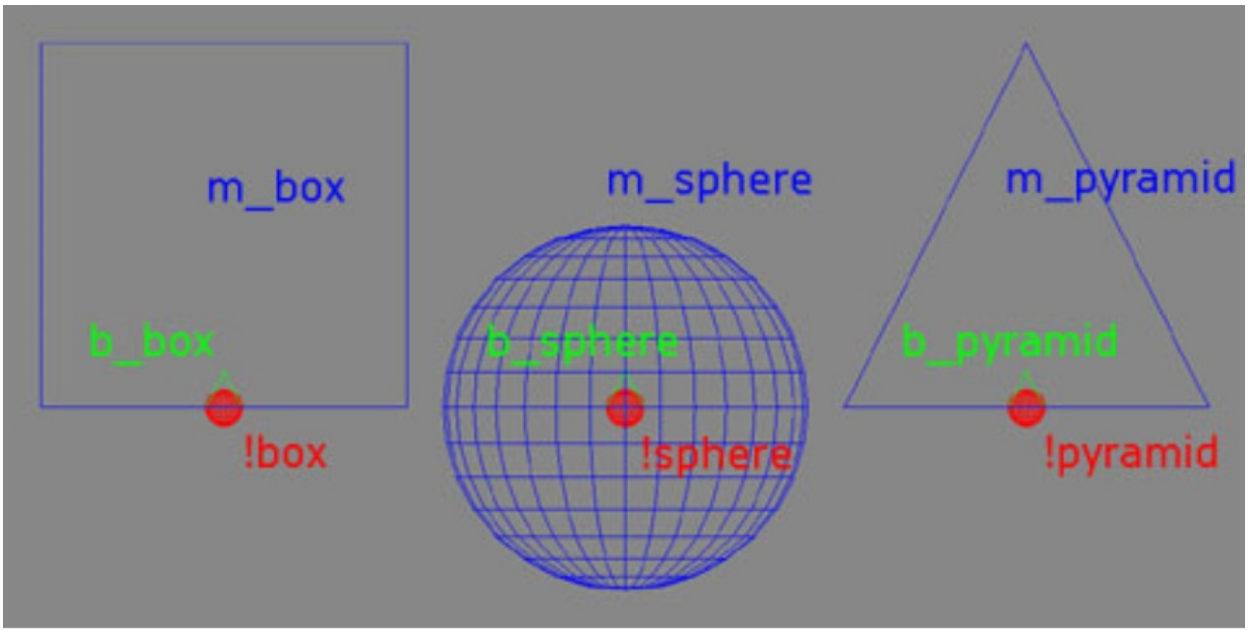


Figure 2 - A 3DS Max file with multiple models correctly named

Once the models have been created, they are exported to a .jmi file. Inside the .jmi are instructions as to where the separate .jms files should be located for each model and part.

> ![Note] You can export a single model from a bulk file by selecting the bang node (named with a !) of the model you want to export, and then choosing **Export Selected** from the file menu in 3ds Max.

Bulk Export from 3DS Max and converting to .jms

Step-by-step bulk file export

1. Create a model in 3ds Max with several separate structures for the Render model, Collision (@ symbol in front of name), and Physics ($ in front of name).
2. Be sure you have a world node (! symbol in front of name) for the object that is named what you want the object to be named after the import.
3. From the 3ds Max File menu, select **Export**.
4. Navigate to the location you want to export the files to and select .FBX as the type.

Convert to .jms using Tool

1. Launch the Tool.
2. Run the following command:

```
tool fbx-to-jms render "[file path to .fbx file]" "[file path to save].jms"
```

- For example: tool fbx-to-jms render " F:,masterchief.fbx" "F:\masterchief.JMS"

There are additional tool command associated with this listed below:

- render,collision,physics-or-all - Sets the type of geo this FBX is for. Use either render, collision, physics, or all only. The geo class set will determine what geometry gets exported from the JMS
- fbx - An absolute filepath to a valid FBX file.
- jms - An absolute filepath that includes name and extension to write the output to.

Bulk Import Using Tool

General information about importing bulk files using Tool (within Guerilla):

- Files are most easily imported using the bulk-import-model-folder command. However, there are a few other tool commands that you may find more useful depending on what you're trying to do. See the bottom of this document for a complete list of commands.
- When a .jmi file is imported, Tool creates render_model, physics_model, collsion_model, and .model tags for the new object. In most cases, you'll have to create the type tag yourself (whether it's .crate, .scenery, etc). See below for a complete list of the bulk import commands and what they do.
- When the files are imported, Tool automatically gives the new tags the same name as the subfolder they are located in.

Step-by-step bulk file import:

- Export your bulk files from 3ds Max.
- Launch Guerilla.
- From the Tools menu, select Run Tool Command (or press shift-ctrl-t).
- Select the bulk-import-model-folder command from the All commands list.
- Go to the Source Directory box of the dialog, click on the ellipses button (...) and choose the location of your .jmi file.
- Click the Run Tool button. The import process will create .render_model, .physics_model, .collsion_model, and .model tags for the new object.

Bulk import tool commands

- **Bulk-render** — Imports all render models from a specified .jmi file.
- **Bulk-physics** — Imports all physics models from a specified .jmi file.
- **Bulk-collision** — Imports all collision models from a specified .jmi file.
- **Bulk-import-model-folder** — Imports all model files from a specified folder. Converts .jms files for render, collision, and physics models and opens or creates .model tag and links everything up correctly.
- **Bulk-import-scenery-folder** — Imports all model files from a specified folder. Converts .jms files for render, collision, and physics models and opens or creates .model tag and links everything up correctly then opens or creates .scenery tag and links everything up correctly.
- **Bulk-import-crates-folder** — Imports all model files from a specified folder. Converts .jms files for render, collision and physics models and opens or creates .model tag and links everything up correctly then opens or creates .crate tag and links everything up correctly.
- **Bulk-import-models** — Imports all model files from a specified .jmi file. Converts specified .jms files for render, collision, and physics models and opens or creates .model tag and links everything up correctly.
- **Bulk-import-scenery** — Imports all model files from a specified .jmi file. Converts .jms files for render, collision, and physics models and opens or creates .model tag and links everything up correctly then opens or creates .scenery tag and links everything up correctly.
- **Bulk-import-crates** — Imports all model files from a specified .jmi file. Converts .jms files for render, collision, and physics models and opens or creates .model tag and links everything up correctly then opens or creates .crate tag and links everything up correctly.

Maya

11/23/2022 • 2 minutes to read

Cinematic Texture Camera Notes

The texture camera allows any bitmap in a shader to be replaced by a real-time rendering of the scene. The placement of the texture camera itself is handled through HS Scripting.

Of course it's expensive.

Set up the texture camera

Any bitmap reference can make use of the texture camera.

Right-click on any bitmap reference then choose extern mode (see Figure 1).



Figure 1 - Extern mode

Once the extern mode has been set, one of the pull-down options is texture camera (see Figure 2).

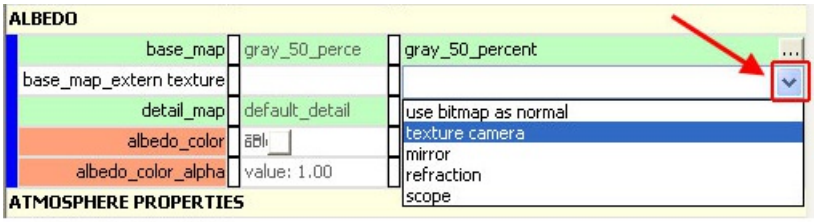


Figure 2 - Texture camera pull-down.

Set up the HS scripting

The texture camera is attached to a named object's marker (the object can be named in Sapien) through HS Scripting.

- texture_camera_set_object_marker <object> <string_id> <real> — Sets the render texture camera to a given object marker
- texture_camera_off — Turns off the render texture camera
- render_debug_texture_camera <boolean> — Toggles displaying the texture camera in the corner of the screen

Additional notes

- This is only expensive because you actually have to render the texture camera

- Once you've rendered it, you can source it everywhere. If you wanted it copied to 100s of displays on a display wall, it's not any more expensive than putting it on one TV and using default bitmaps on the rest
- Aspect ratio can't be set — it always assumes square pixels and a square bitmap.
- When you map the texture, it needs to be mapped in the ratio 640×480, but once you have that mapped you can crop it however you want to fit your monitor, screen, etc.
- You need to set the scale of the shader as well. So you need to right-click and put in base_map_scale_x and base_map_scale_y and enter those values

Animation

11/23/2022 • 2 minutes to read

Facial Animation

Information on the Halo 3 facial animation system.

Naming Conventions

Naming conventions used for animation files.

Vehicle Animation

Information and breakdowns on Vehicle animations.

Facial Animation

11/23/2022 • 4 minutes to read

You can have eight emotional states which may seem limiting, but they can be blended to create more than 64 individual emotional states. For example, the canonical 7 emotions identified by FACS can be blended to create 47 emotions.

> **IMPORTANT**
> This section is built from the original documentation and is tailored for use in Maya, but the important principles can be used in any 3d program.

Generating the Animation

Each character that wishes to speak or show emotion must contain two animations in its graph:

- *facial_animation_base.jmr*
- *facial_animation_overlay.jmo*

These animations are made from the same Maya file, exported twice. Once as a replacement animation to create *facial_animation_base.jmr* and once as an overlay to generate *facial_animation_overlay.jmo*.

> **NOTE**
> Different frame counts are exported for each animation. The total frame set to author is shown below:

- Base Pose— Matching models default pose
- Base Pose Copy— Must match the base pose!
- Base Pose Blink— Must match the base pose, except for eye blink

Neutral Face

- Neutral: Silent— Mouth closed.
- Neutral: Eat
- Neutral: Earth
- Neutral: If
- Neutral: Ox
- Neutral: Oat
- Neutral: Wet
- Neutral: Size
- Neutral: Church
- Neutral: Fave
- Neutral: Though
- Neutral: Told
- Neutral: Bump
- Neutral: New

- Neutral: Roar
- Neutral: Cage
- Neutral: unused
- Neutral: Eyebrow Raise— Must match r;silent pose, except for eyebrows
- Neutral: Eyebrow Lower— Must match r;silent pose, except for eyebrows
- Neutral: Blink— Must match r;silent pose, except for blink

Happy Face

- Happy: Silent— Mouth closed.
- Happy: Eat
- Happy: Earth
- Happy: If
- Happy: Ox
- Happy: Oat
- Happy: Wet
- Happy: Size
- Happy: Church
- Happy: Fave
- Happy: Though
- Happy: Told
- Happy: Bump
- Happy: New
- Happy: Roar
- Happy: Cage
- Happy: unused
- Happy: Eyebrow Raise— Must match silent pose, except for eyebrows
- Happy: Eyebrow Lower— Must match silent pose, except for eyebrows
- Happy: Blink— Must match silent pose, except for blink

Sad Face— Frames 43–62, repeat the phoneme set for a Sad character

Angry Face— Frames 63–82, repeat the phoneme set for an Angry character

Disgusted Face— Frames 83–102, repeat the phoneme set for a Disgusted character

Scared Face— Frames 103–122, repeat the phoneme set for a Scared character

Surprised Face— Frames 123–142, repeat the phoneme set for a Surprised character

Pain Face— Frames 143–162, repeat the phoneme set for a character in Pain

163–165— Unused (should match base pose of frame #0)

166— Orientation Head Pitch Pos

167— Orientation Head Pitch Neg

168— Orientation Head Roll Pos

169— Orientation Head Roll Neg

170— Orientation Head Yaw Pos

171— Orientation Head Yaw Neg

172— Gaze Eye Pitch Pos

173— Gaze Eye Pitch Neg

174— Gaze Eye Yaw Pos

175— Gaze Eye Yaw Neg

176— Eyebrow Raise Left

177— Eyebrow Lower Left

178— Eyebrow Raise Right

179— Eyebrow Lower Right

Animation Graphs

1. First, export the entire 180 frame animations as **facial_animation_overlay.jmo**.
2. Next, export only the first 162 frames (all the emotions and phonemes) as **facial_animation_base.jmr**.
3. Reimport the character's animation graph so that the new animations are included.
4. Make sure compression is disabled for both (Best Accuracy) and save the graph.

Scripted Emotions

There are two ways to instruct a unit to show emotion: scripting and lipsync.

In scripts, we provide the command:

```
unit_set_emotion_by_name \<unit> \<emotion> \<weight> \<seconds>*
```

This command tells <unit> to become <emotion> over the next <seconds> seconds. The amount in which they display the emotion is indicated by <weight>. For example, *unit_set_emotion_by_name johnson happy 0.5 3* instructs the character **Johnson** to display his happy face at 50% strength. If Johnson is not already showing his happy face at 50%, he can use the next 3 seconds to interpolate to it.

Emotions can be combined to create new emotions or cross-fade between two or more emotions. If you had already used the happy example above, and then added unit_set_emotion_by_name johnson angry 0.5 3 the r;johnson character would interpolate to show both the happy and angry faces at the same time, each with the corresponding weight. To shut off an emotional face, simply give it a weight of zero to interpolate to: unit_set_emotion_by_name johnson angry 0 1

In this example, the angry face would shut off over 1 second. Specifying a time value of zero will make the change instant.

Embedded Emotions

We can also embed emotional changes directly into the lipsync animations. You can do this by either hand-editing an animation and exporting an fxx file, or adding emotional commands to the text file associated with your audio.

For example, imagine we had a sound file (angry_marine.aif) and a text file (angry_marine.txt) containing the dialog:

Enemies spotted, kill them!

We could embed emotional response into the text file, and have it affect the animation produced.

["r;shock" 50%] Enemies spotted, [/shock] ["r;anger" 100%] kill them! [/anger]

This would instruct the system to add the shocked emotional state to the two three words of the dialog at 50% strength, and 100% of anger for the final two words.

> **NOTE**
> The actual parameters used are more complicated than the ones shown in this example and provide full control over the weighting curve generated.

Further Notes

During a cinematic the animators have full control over the face until one of the following happens:

- a script command sets an emotion
- dialog begins

When one of these happens the face nodes moved by the facial_animation_base animation will transition to engine control. Even if the animator has moved these nodes, they will be trumped by the runtime system. Once the dialog ends, or the script clears the emotion, those nodes revert back to animator control.

Nodes which are not moved by facial_animation_base (usually the head & eyes) will always remain under the animator's control.

There is no way to turn phoneme animation on or off, it turns itself on when dialog audio is running and shuts down when it isn't.

Vehicle Animation

11/23/2022 • 7 minutes to read

This article contains descriptions of the various types of vehicle animations used in the Halo engine and instructions on how to set up each one. All vehicles can be set up to use the animations described below— but vehicles don't need to have any of them to function in the engine.

> **IMPORTANT**
> This section is built from the original documentation and is tailored for use in 3DS Max, but the important principles can be used in any 3d program.

General Instructions

- Only place keyframes on nodes you plan to animate
- Export all animations to an animations directory for each vehicle
- All animations should be exported as .jmo files
- Create one 3ds Max file and .jmo for each animation

Vehicle Animations

Halo 3 has nine distinct classes of vehicles (10 if you include turrets), each with their own set of animations.

Common Animations

All vehicles may use these animations regardless of their class.

- *Suspension* — A vehicle may have any number of suspension animations setup in its animation graph
 - Example: suspension:left:back — .jmo animation, shows the tire or suspension part traveling from full compression to full extension. Additional data entry is also needed in the Vehicle Suspension section of the graph
- *Acceleration Screens* — Any vehicle may use acceleration screens to depict its reaction to accelerating. Typically only used for secondary motion on mounted turrets
 - Example: combat:acc_front_back — .jmo animation, shows the vehicle accelerating from one axis extreme to another
- *Opening and Closing* — Any vehicle may open and close as an operator enters and leaves. A vehicle is open when empty and closed when an operator is inside. Therefore, when a player gets into a vehicle it may close around him and then later open to let him out. Unmanned vehicles then remain open on the battlefield waiting for a new operator
 - Example 1: combat:closing — .jma animation, shows the vehicle closing around a driver. The last frame will be held to keep the vehicle closed
 - Example 2: combat:opening— .jma animation, shows the vehicle opening around to release a driver. The last frame will be held to keep the vehicle open

Prowler

- *Steering* — Shows the vehicle turning left or right
 - Example: combat:steering— .jmo animation, shows the vehicle turning left to right

Ghost

- *Steering* – Shows the vehicle turning left or right
 - Example: combat:steering— .jmo animation, shows the vehicle turning left to right

Chopper

- *Steering* – Shows the vehicle turning left or right
 - Example: combat:steering — .jmo animation, shows the vehicle turning left to right
- *Ground Speed* — Shows the vehicle traveling forward
 - Example: vehicle:ground_speed— .jmo animation, shows the vehicle moving forward (usually one revolution of tires or treads).

Hornet

- *Lift Aim Screens* — Aim screens for the left and right turbines
 - Example 1:combat:l_lift_aim — .jmo animation, aim screen for left turbine
 - Example 2:combat:r_lift_aim— .jmo animation, aim screen for right turbine

Warthog

- *Steering* — Shows the vehicle turning left or right
 - Example: combat: steering — .jmo animation, shows the vehicle turning left to right
- *Ground Speed* — Shows the vehicle traveling forward
 - Example: vehicle:ground_speed — .jmo animation, shows the vehicle moving forward (usually one revolution of tires or treads)
- *Ground Speed Rear*— Shows the rear of the vehicle traveling forward (for dual axle vehicles)
 - Example: vehicle:ground_speed— .jmo animation, shows the vehicle moving forward (usually one revolution of tires or treads)

Vehicle Marker Keyframe Naming Conventions

Vehicle markers placed for biped hand and foot IK follow this naming convention:

- #(seat name)_(marker_name)_(optional biped name)

For example:

- mauler_p_l_left_foot_brute
- wraith_g_left_hand (if all characters use the same hand position)

Also, seat names are based on the vehicle and seat type and are the same as the marker the seat animations are exported relative to.

Vehicle Suspension Animation

To author suspension animation for a vehicle, you must determine the compression and extension of the tire required to make the animation function correctly.

First you need to author the animation. It's a simple three frame animation with the following:

- **Frame 0** — Rest position
- **Frame 1** — Full extension

- **Frame 2** — Full compression

Now determine the proper compression data for the tire— otherwise your animation won't work. See the steps below to do this...

Step 1

Determine the rest position height from the ground. This is done by measuring the distance from the ground to the center of the tire marker. (typically, tire markers are located at the origin of the tire).

As you can see in Figure 1, the front right tire of the Mongoose is 14 Max units above the ground plane— .14 world units (or wu).

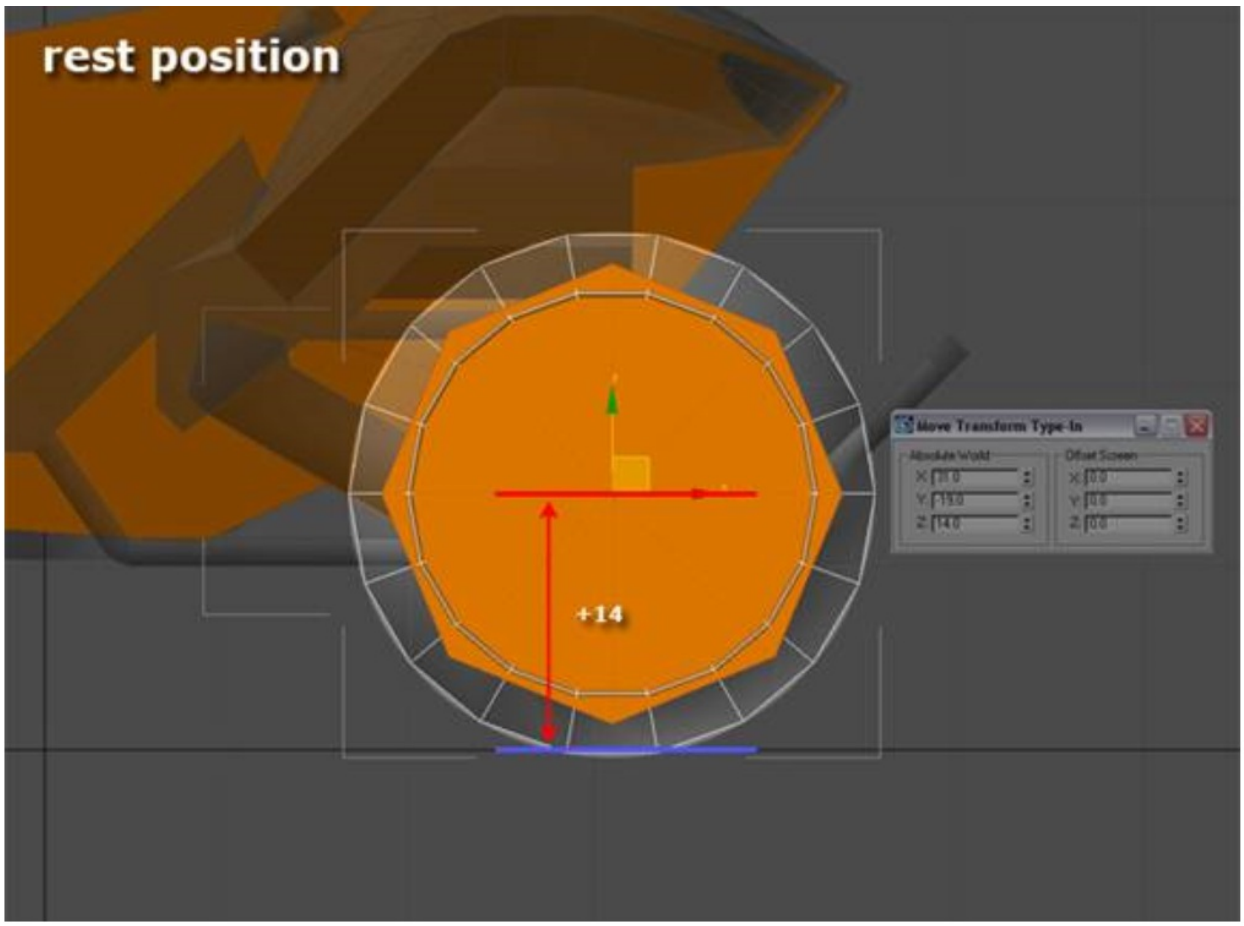


Figure 1 - Rest Position.

Step 2

Determine the full extension of the tire:

1. Move to frame 1 of your animation and select the tire.
2. Choose a vert on the bottom of the tire.
3. Take the distance of this vert from the ground plane (for example, 8.372 Max units) and add this to the rest position value (e.g. 14), then take that value and multiply by .01 to get the world unit value. In this example, the full extension of the tire is .2237 wu from the rest position.

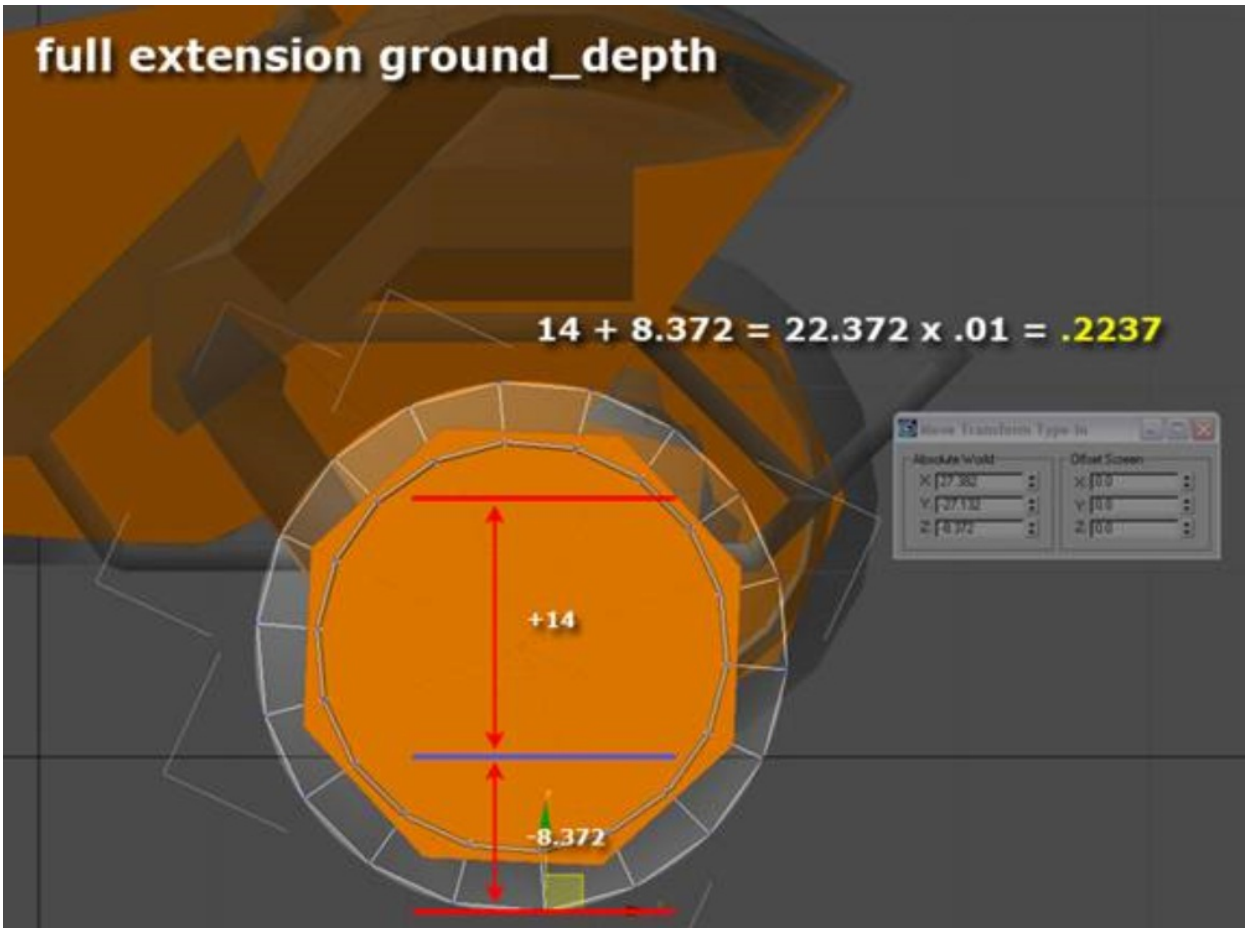


Figure 2 - Full Extension.

Step 3

Determine the full compression of the tire:

1. Move to frame 2 of your animation.
2. Select the tire and choose a vert on the bottom of the tire.
3. Take the distance of this vert from the ground plane (for example 10.842 Max units) and subtract this from the rest position value (e.g. 14), then take that value and multiply by .01 to get the world unit value. In this case, the full compression of the tire is .0315 wu from the rest position.

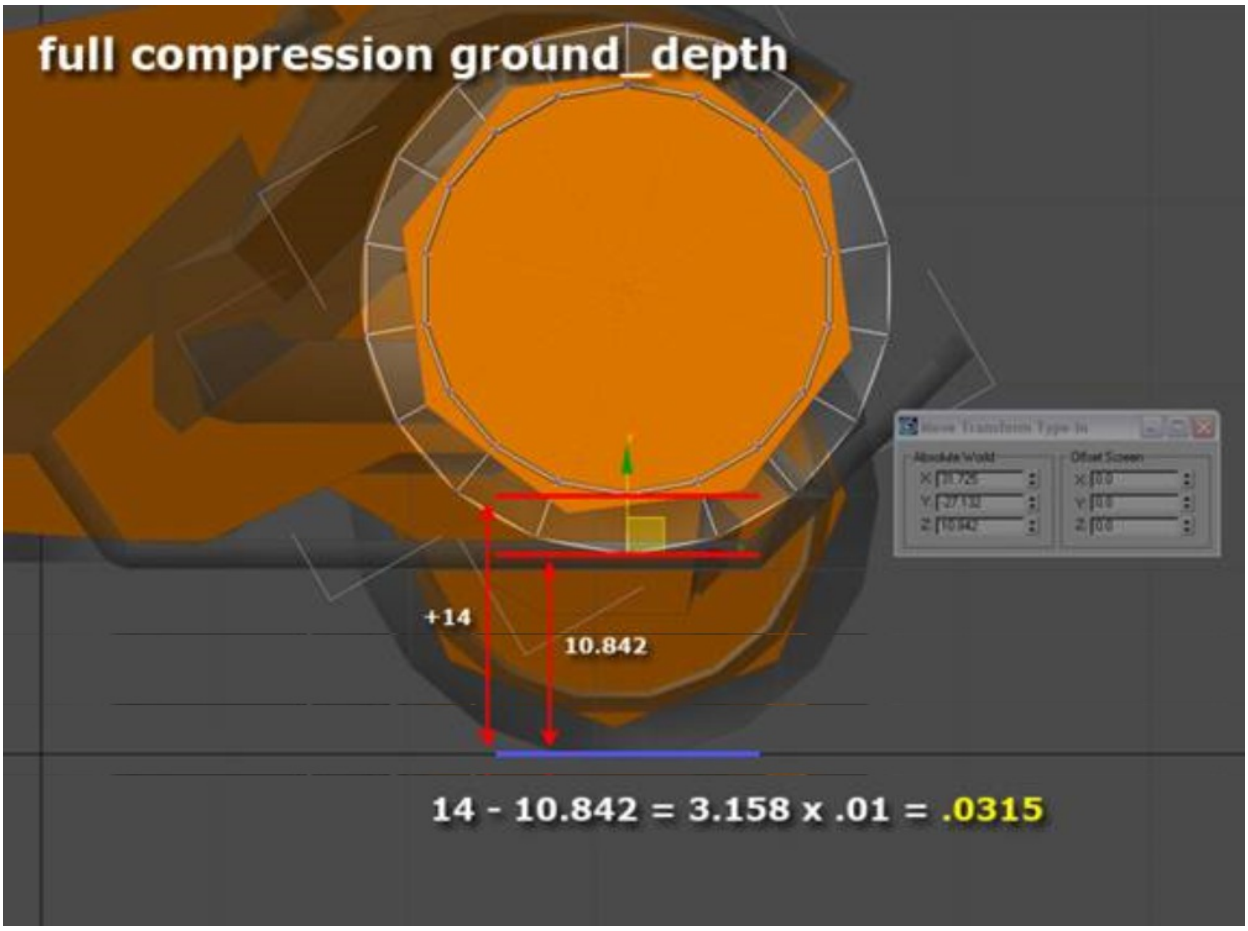


Figure 3 - Full Compression.

Step 4

After you have those two values, you need to plug them into the vehicle's animation graph. Scroll down to the vehicle suspension block to find this data. Enter the values in the appropriate fields circled in Figure 4.

You also need to input the marker name for that tire so the code has a proper reference point for the true rest position of the tire.

After you have completed this, you will need to calculate the values for the other tires.

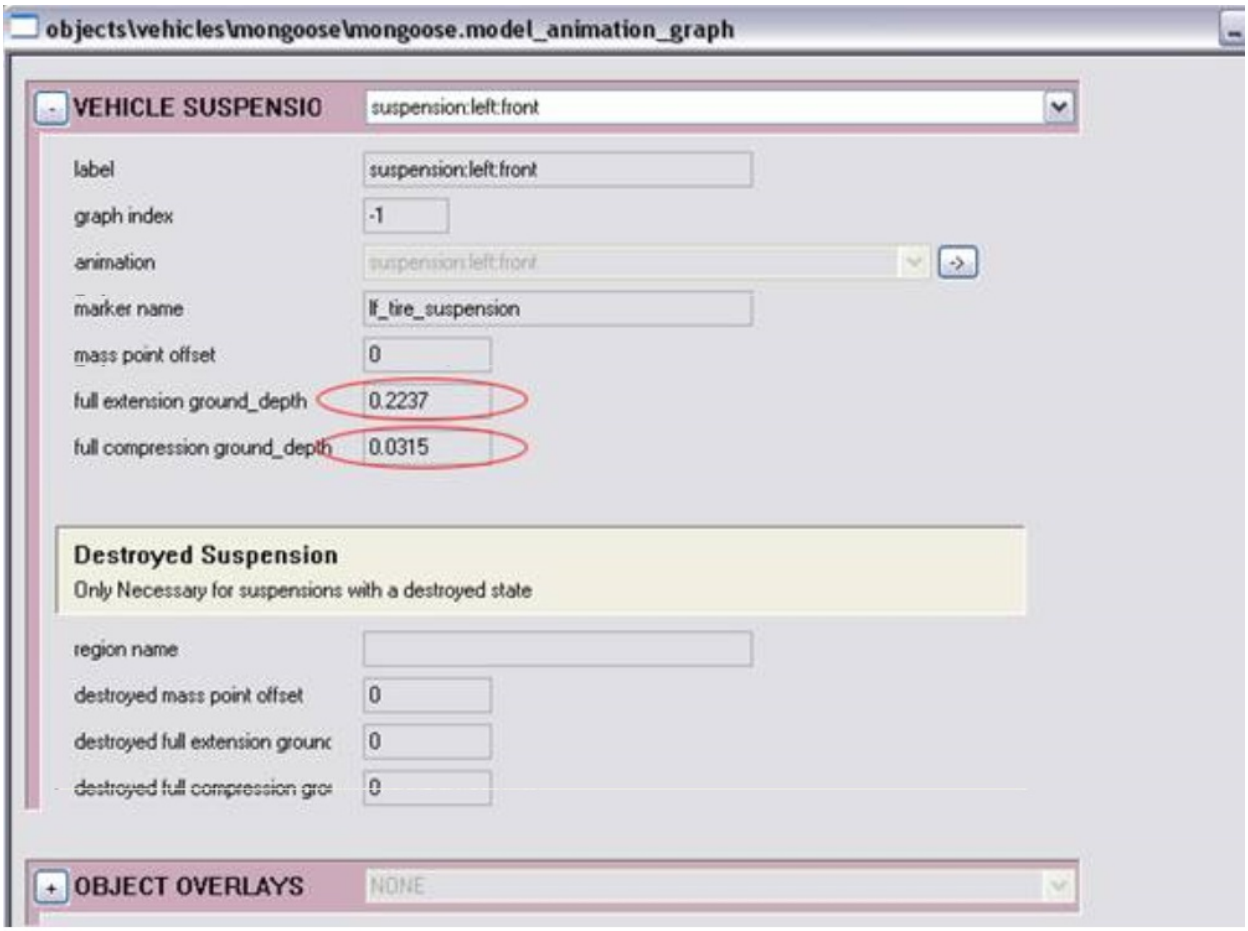


Figure 4 - Vehicle Animation Graph.

Vehicle Seat Names

VEHICLE	SEAT	SEAT NAME
Banshee	Driver	banshee_d
Banshee	Boarding D	banshee_b_d
Banshee	Boarding Left	banshee_b_l
Banshee	Boarding Right	banshee_b_r
Banshee	Boarding Front	banshee_b_f
Brute Chopper	Driver	ghost_d
Brute Chopper	Boarding	ghost_b_l
Brute Chopper	Boarding	ghost_b_b
Brute Chopper	Boarding	ghost_b_g
Brute Chopper	Boarding	ghost_b_h
Brute Hovercraft	Driver	warthog_d

VEHICLE	SEAT	SEAT NAME
c_turret_ap	Gunner	c_turret_ap_d
Cov_Guntower	Driver	c_turret_ap_d
Creep	Driver	creep_d
Creep	Gunner	creep_g
Creep	Passenger Left	creep_p_l01
Creep	Passenger Left	creep_p_l02
Creep	Passenger Left	creep_p_l03
Creep	Passenger Left	creep_p_l04
Creep	Passenger Left	creep_p_l05
Creep	Passenger Right	creep_p_r01
Creep	Passenger Right	creep_p_r02
Creep	Passenger Right	creep_p_r03
Creep	Passenger Right	creep_p_r04
Creep	Passenger Right	creep_p_r05
Creep	Small Cargo	creep_sc01
Ghost	Driver ghost_	d
Ghost	Boarding	ghost_b_l
Ghost	Boarding	ghost_b_h
Ghost	Boarding	ghost_b_g
Ghost	Boarding	ghost_b_b
Gravity Throne	Driver	throne_d
Gravity Throne	Boarding left	throne_b_l
Gravity Throne	Boarding right	throne_b_r
h_turret_ap	Driver/Gunner	h_turret_ap_d
Insertion Pod	Passenger	insertion_pod_d

VEHICLE	SEAT	SEAT NAME
Longsword	Driver	longsword_d
Mongoose_x	Driver	mongoose_d
Mongoose_x	Passenger	warthog_p
Pelican	Driver	pelican_d
Pelican	Passenger	pelican_p_l01
Pelican	Passenger	pelican_p_l02
Pelican	Passenger	pelican_p_l03
Pelican	Passenger	pelican_p_l04
Pelican	Passenger	pelican_p_l05
Pelican	Passenger	pelican_p_r01
Pelican	Passenger	pelican_p_r02
Pelican	Passenger	pelican_p_r03
Pelican	Passenger	pelican_p_r04
Pelican	Passenger	pelican_p_r05
Pelican	Gunner	pelican_g
Pelican	Large Cargo	pelican_lc
Pelican	Entry	pelican_e
Phantom	Driver	phantom_d
Phantom	Large Cargo	phantom_lc
Phantom	Creep	phantom_lc_creep
Phantom	Passenger	phantom_p_a01
Phantom	Passenger	phantom_p_a02
Phantom	Passenger	phantom_p_a03
Phantom	Passenger	phantom_p_b01
Phantom	Passenger	phantom_p_b02

VEHICLE	SEAT	SEAT NAME
Phantom	Passenger	phantom_p_b03
Phantom	Passenger	phantom_p_c01
Phantom	Passenger	phantom_p_c02
Phantom	Passenger	phantom_p_c03
Phantom	Small Cargo	phantom_sc01
Phantom	Small Cargo	phantom_sc02
Scorpion	Driver	scorpion_d
Scorpion	Passenger (right-front)	scorpion_p_rf
Scorpion	Passenger (right-back)	scorpion_p_rb
Scorpion	Passenger (left-front)	scorpion_p_lf
Scorpion	Passenger (left-back)	scorpion_p_lb
Scorpion	Boarding (driver-left)	scorpion_b_d_l
Scorpion	Boarding (driver-right)	scorpion_b_d_r
Scorpion	Boarding (back)	scorpion_b_b
Spectre	Driver	spectre_d
Spectre	Passenger (left)	spectre_p_l
Spectre	Passenger (right)	spectre_p_r
Spectre	Boarding (driver)	spectre_b_d
Spectre	Boarding (passenger-left)	spectre_b_p_l
Spectre	Boarding (passenger-right)	spectre_b_p_r
VTOL	Driver	warthog_d
VTOL	Boarding	banshee_b_d
VTOL	Boarding	banshee_b_f
VTOL	Boarding	banshee_b_l
VTOL	Boarding	banshee_b_r

VEHICLE	SEAT	SEAT NAME
Warthog	Driver	warthog_d
Warthog	Passenger	warthog_p
Warthog	Boarding	warthog_b_d
Warthog	Boarding	warthog_b_p
Warthog	Boarding	warthog_b_b
Warthog	Boarding	warthog_b_h
Wraith	Driver	wraith_d
Wraith	Boarding	wraith_b_l
Wraith	Boarding	wraith_b_r

Vehicle Variant Names

Below are lists of all of the variant names for each vehicle in Halo 3.

VEHICLE NAME	VARIANT
banshee	default
	damaged
	heretic
brute_chopper	NA
brute_hovercraft	NA
c_turret_ap	default
capital_ship	NA
civilian\bus	blue_green_bus
	green_white_bus
	tan_orange_bus
	yellow_black_bus
civilian\hotrod	NA
civilian\mlx	dark_blue_mlx

VEHICLE NAME	VARIANT
	red_mlx
	black_mlx
	yellow_mlx
	white_mlx
	dark_blue_mlx_destroyed
civilian\new\car_all_purpose	default
	truck
civilian\new\forklift	NA
civilian\new\truck_cab_large	NA
civilian\panel_truck	red_panel_truck
	white_panel_truck
	brown_panel_truck
	panel_truck_destroyed
civilian\piglet	NA
civilian\streetbike	NA
civilian\uberchassis	default
	destroyed
	burning
	fuxored
creep	default
	cargo
ghost	default
	damaged
gravity_throne	NA
h_turret_ap	default

VEHICLE NAME	VARIANT
insertion_pod	default
	open
longsword	NA
mongoose_x	NA
pelican	default
	attack
	unarmed
pelican_x	default
phantom	default
	noturret
	long_lift
scorpion	default
	damaged
	multiplayer
spectre	default
	damaged
vtol	default
warthog	default
	troop
	damaged
	gauss
warthog_x	NA
wraith	default
	damaged
	multiplayer

Naming Conventions

11/23/2022 • 2 minutes to read

This article discusses the current name conventions for animation files.

- Animation Prefix: Every animation name begins with a mode, weapon class, and weapon type
 - Mode - combat, patrol, etc.
 - Weapon Class - rifle, pistol, etc.
 - Weapon Types are abbreviated versions of the weapon name
 - Examples: ar (assault rifle), ne (needler), pp (plasma pistol), etc.
 - Format: <mode> <weapon class> <weapon type>
 - Note: The word 'any' can be used to replace a weapon class and/or weapon type, indicating the animation can be used for any weapon class and/or weapon type
 - Example: combat pistol any or combat any any
- After the prefix, the animation name contains either a State, Death and Damage, or a State and Transition
 - State - idle, move-left, move-front, etc.
 - Format: <mode> <weapon class> <weapon type> <state>
 - Example: combat rifle ar idle
 - Death and Damage is comprised of damage type, direction, and region
 - Damage Type - h_ping, s_ping, h_kill, s_kill
 - s (soft): refers to damage done by something like a pistol
 - h (heavy): refers to damage done by something like a rocket launcher
 - ping: refers to being hit
 - kill: refers to being killed
 - Direction refers to the direction from which the damage is being inflicted
 - left, right, front, back
 - Region refers to the region on the character that is being damaged
 - gut, chest, head, l_arm, l_hand, l_leg, l_foot, r_arm, r_hand, r_leg, r_foot
 - Format: <mode> <weapon class> <weapon type> <damage type> <direction> <region>
 - Example: combat rifle ar s_ping front gut
 - Transitions refer to animations in which the character is changing modes or states and is denoted by placing a '2' in the name, followed by the new mode and state.
 - Format: <mode> <weapon class> <weapon type> <state> 2 <mode> <state>
 - Example: combat rifle ar idle 2 combat patrol
- NOTE: With regards to files that contian 'any', some files may have omitted the word and left the space

blank (eg. combat rifle idle). In such cases, 'any' is implied. Check with your lead for how they prefer to handle this scenario.

- File Extensions
 - .jma – extracts root motion in the XY plane, and zeros out any vertical motion. No orientation
 - .jmm- extracts no movement
 - .jmt – extracts root motion in the XY plane and yaw orientation (used for turns, for example)
 - .jmo – no movement (these are overlays)
 - .jmr – no movement (replacement animation in object space)
 - .jmrx – no movement (replacement animation in local space)
 - .jmz – extracts FULL XYZ motion and yaw orientation (could not find examples of this)
 - .jmw – animation in world space (used for large objects that cross large areas of a level)

Decal

11/23/2022 • 2 minutes to read

System

System

11/23/2022 • 4 minutes to read

Decals are graphics placed in maps and are often used for decorative features like bullet holes, cracks, and signs.

Decals are textures that are applied directly to shapes. Before decals, the way to apply, for example, a sign to a wall was to make a distinct object with the texture of a sign and then place that object very very close to the surface of the wall. This is a very an expensive way to put a sign into a map. Decals are free-floating textures that apply themselves to the surface of an object instead of being a distinct object in the level. Decals are inexpensive by comparison.

Place a Decal

1. Open Sapien.
2. Open the scenario you plan to edit.
3. Use Decal Viewer to find the decal you want.
4. In Sapien, go to the Hierarchy view and expand Game Data.
5. Click Decals.

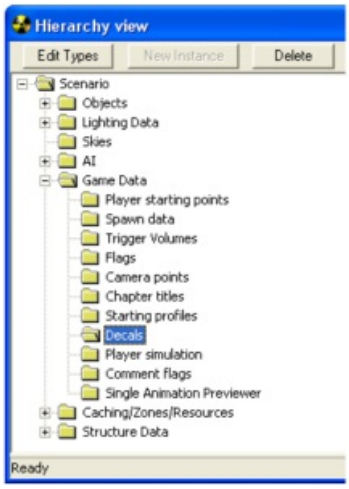


Figure 1 - Hierarchy View.

6. Click the Edit Types button (See Figure 1).
7. Ensure that decal_system is selected in the Object Class pulldown (see Figure 2).

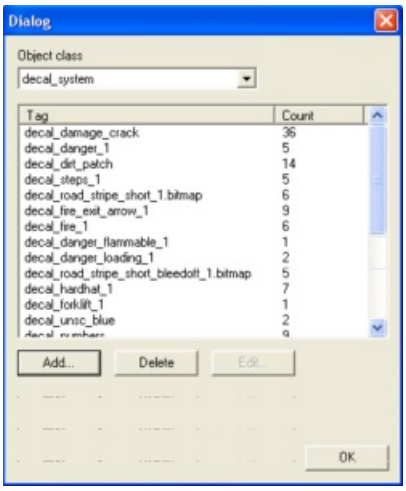


Figure 2 - Edit Types Dialog.

> **NOTE**
> Count indicates how many of that decal have been placed on the map.

8. Click Add.
9. Navigate to your decal using the path noted in Decal Viewer.
10. Click Add Tag. **NOTE: Only click this button once**.
11. Click Done.
12. Click OK in the dialog box that appears.
13. Right-click in the game window where you would like the decal to appear.

> **NOTE**
> Keep an eye on the Output window to make sure you're not placing too many decals.

14. Save your scenario.
15. Press ctrl+shift+f to see the decals directly in Sapien.

It's easier to place decals if the checkbox only draw selected items is checked (see Figure 3). This will hide all the gizmos of the decals in the level. Hover over a decal to show its gizmo.

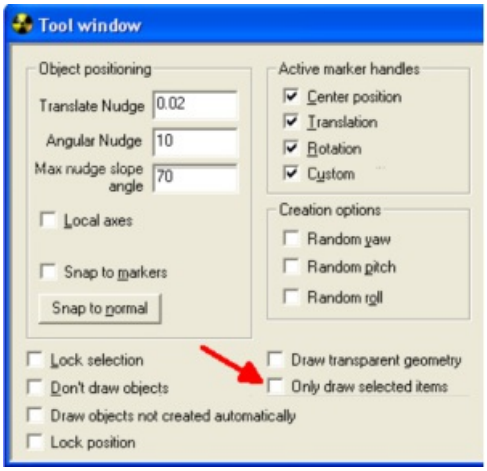


Figure 3 - Only Draw Selected Items option.

Decal gizmo

- White gizmo— Placement (see Figure 4). Click and drag the white box to move the decal.
- Blue gizmo— Rotation (see Figure 4). Click and move the blue box to rotate the decal. Rotation defaults to 45° increments— hold the alt key to free rotate.



Figure 4 - Decal Gizmo.

Notes

- **Important**: The delete key will delete your decal— there is no undo.
- If you see a black rotating square in the upper left corner of your map, this is an error message that indicates that the decal is too expensive for the level and won't display correctly.
- Curves are the most expensive surfaces to apply decals onto so take care when you're placing decals. Use cull angle and clamp angle (see below) to reduce this expense.
- There's a limit to the number of decals a map can have—this number varies from map to map.
- If only draw selected items is checked, hover your mouse over a decal until the gizmo appears to adjust the decal.
- If you lose track of a decal, double-click its name in the hierarchy view. Decals are listed in the order they were placed.
- If your decal ghosts, this is a bug. The only way to fix it is to reset the map (press alt+r).
- To experiment with a decal, Avalanche is a good map to practice on.

Vector decals

Use vector decals when you need scalability with crisp edges.

Creating decals

When creating or editing decals, there are some tags that require further explanation.

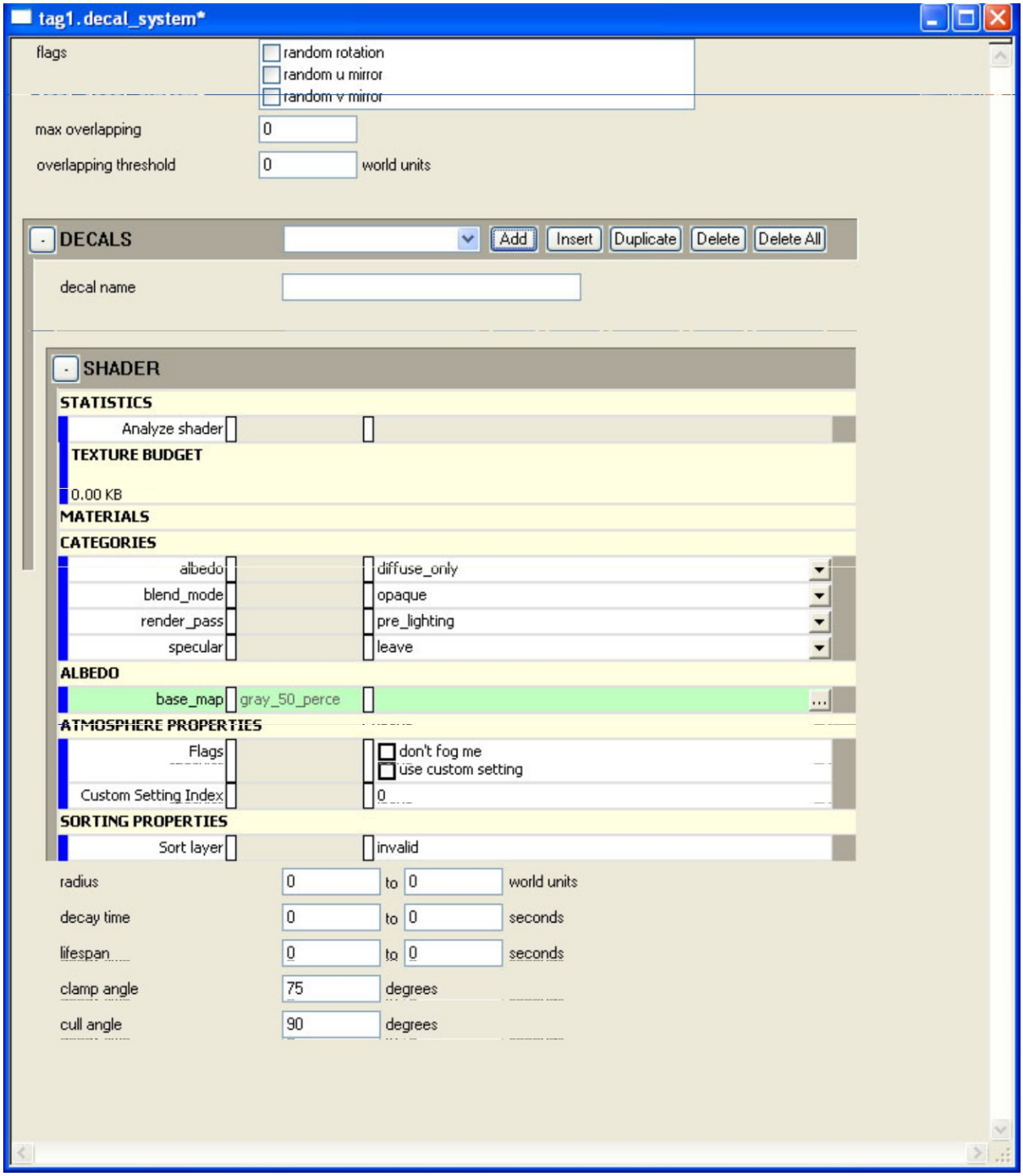


Figure 5 - Decal System.

The following items from Figure 4 are relevant to decals:

Decals

- **decal name**— Name of the decal.

Shader

- **blend mode**— Multiply adds darkness to the decal, Additive adds light to the decal, and Alpha forces the decal to use just the alpha channel for the decal.
- **specular**— Experiment with this tag to see if the decal looks better or worse after applying this item.

Sorting Properties

- **Radius**— The range the decal will expand or shrink to. Some decals will resize according to the surface they're applied onto. This variable affects the limits of how small or large that radius will be. A good rule of thumb is to pick a range of 20 to 30.
- **Decay time**— Used for special effects.
- **Lifespan**— Used for special effects.
- **Clamp angle**— If a decal is applied to an angled surface the decal may stretch to fill the surface, thereby looking terrible. Clamp angle sets the angle at which the decal will not stretch.
- **Cull angle**— If a decal covers a surface that has two faces, say the top and side of a cube, this value sets the angle at which the decal will not continue its coverage. For example, if the cull angle was set to 90° and the decal was placed on top of a cube, the decal wouldn't spread out over the side of the cube and only cover the top of the cube.

Decorators

11/23/2022 • 2 minutes to read

Decorators are a special type of object that are relatively inexpensive to render and have no collision. Decorators are often used for small plants, but can also be used to add clutter to an area or add skybox like elements in the distance.

There are three articles concerning decorators:

Modeling

Texturing

Painting



Figure 1 - Example of Decorators.

Modeling

11/23/2022 • 2 minutes to read

Decorators have several unique quirks related to their modeling:

- The Center of the model is the origin for the decorator
- All decorators must be part of a set
- Each set can have only one material type
- All decorators in a set should have a similar number of verts
- No scaling should be present on a decorator

Decorator Heirarchy

The general heirarchy of a decorator file for use with the decorator export script is as follows:

```
!decorator_set_name1
  b_node
    decoratorN
```

```
!decorator_set_name2
  b_node
    decoratorN
```

```
!decorator_set_nameN
  b_node
    decoratorN
```

Where decorator_set_name is the name of the decorator set and decoratorN represents any number of decorators. See Figure 1 for the heirarchy of a max file.

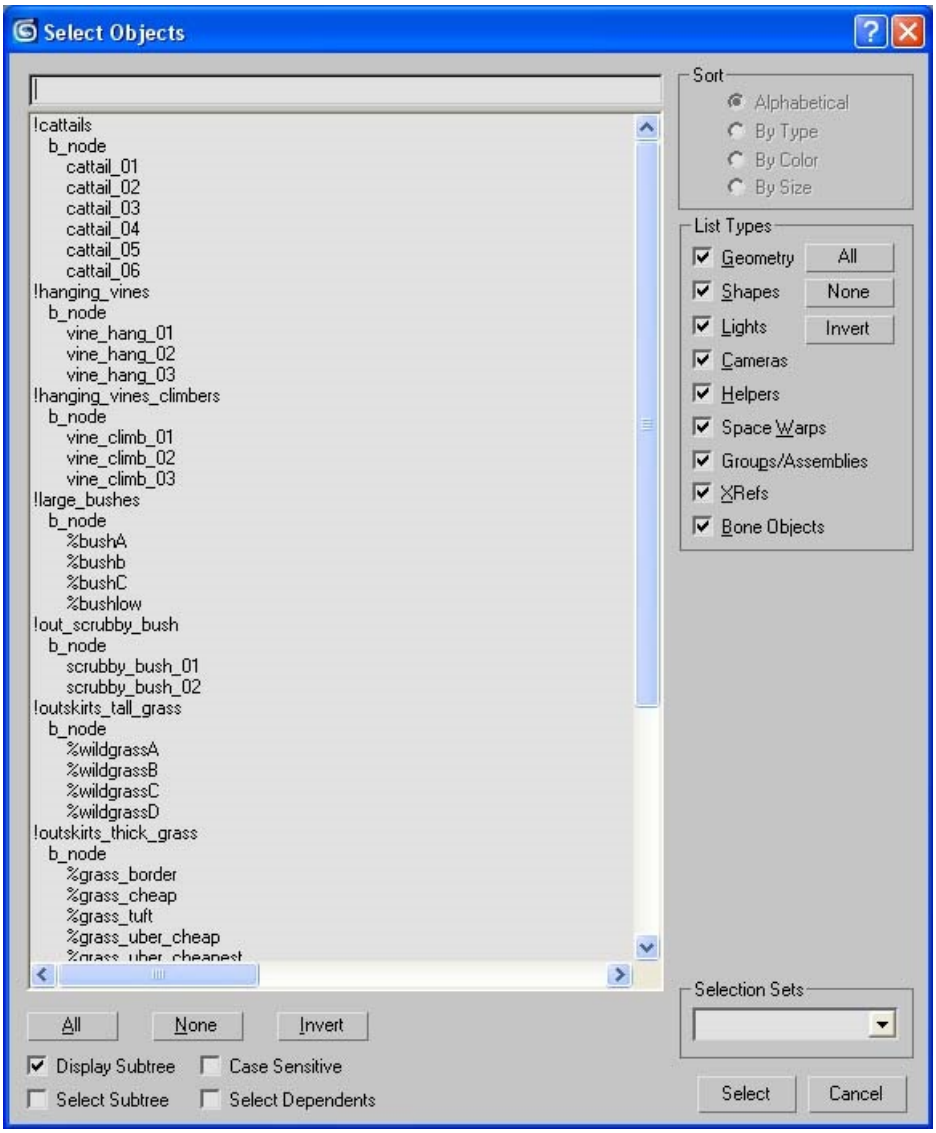


Figure 1 - Decorator File Heirarchy.

Each ! node will create a decorator set and render model as well as an ASS file. Though not needed, the only character that should be present on meshes is % preceding the decorator name.

Model Orientation

Decorators can work in several ways.



Figure 2 - Model Orientation.

There are three instances of orientation. Decorators can:

- Maintain their max alignment when painted. This is most often used for decorators that need to maintain a vertical orientation, such as cat tails or hanging vines. Figure 2 shows a hanging vine. It's designed to appear to be hanging straight down affected by gravity and should not change its rotation with the surface it's placed on.
- Align themselves to the normal of whatever they are painted on. This instance is fairly common and is used for any leaf or ground cover— most grasses and bushes work this way.
- Be set to "hang" against a surface. This instance is used for specialized applications such as the wire on wall decorator (see Figure 3).

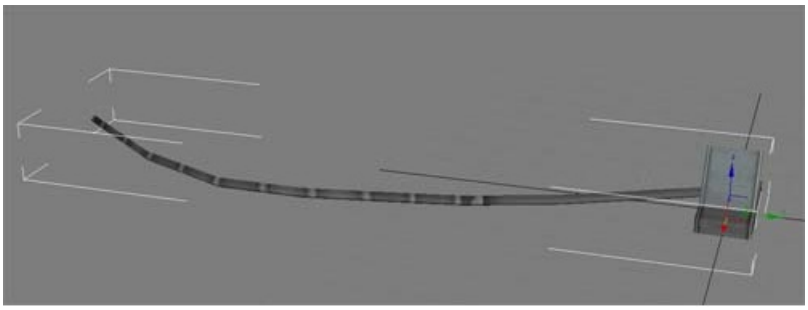


Figure 3 - Wire on Wall Decorator.

Light Sampling

Decorator sample for lights from the center of their bounding box, which encompasses all vertexes in the entire decorator set. This means that decorators in a set should be of similar size as well as similar vertex count

Painting

11/23/2022 • 6 minutes to read

Painting decorators involves using Sapien to place the decorators in a map and requires the decorators resource for forked scenarios. Adding decorator sets is slightly different from the typical means of adding objects to a scenario.

Adding Decorator Sets in Saipien

For each decorator set you want to use in your scenario, you will have to create a new instance in the Scenario\Structure Data\Decorator brush\Decorator sets folder (see Figure 1).

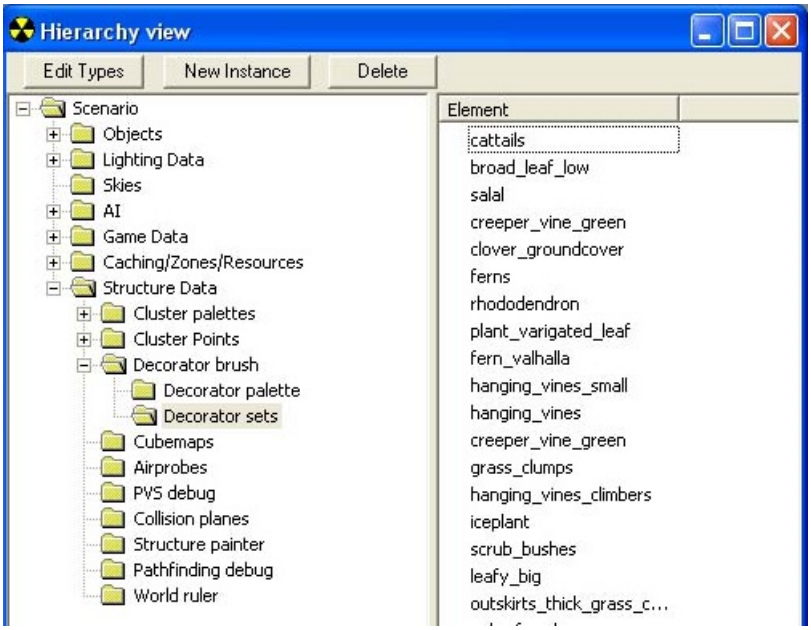


Figure 1 - Decorator Sets Path.

Decorator Brush Tool

The Decorator Brush tool is used to place and edit individual decorators on the map. To use the tool you will first need to select a decorator set, a decorator palette, or choose the select all decorator sets option. After you have selected decorators to edit you can then use the actions available to be bound to the left and right mouse buttons (see Figure 2).

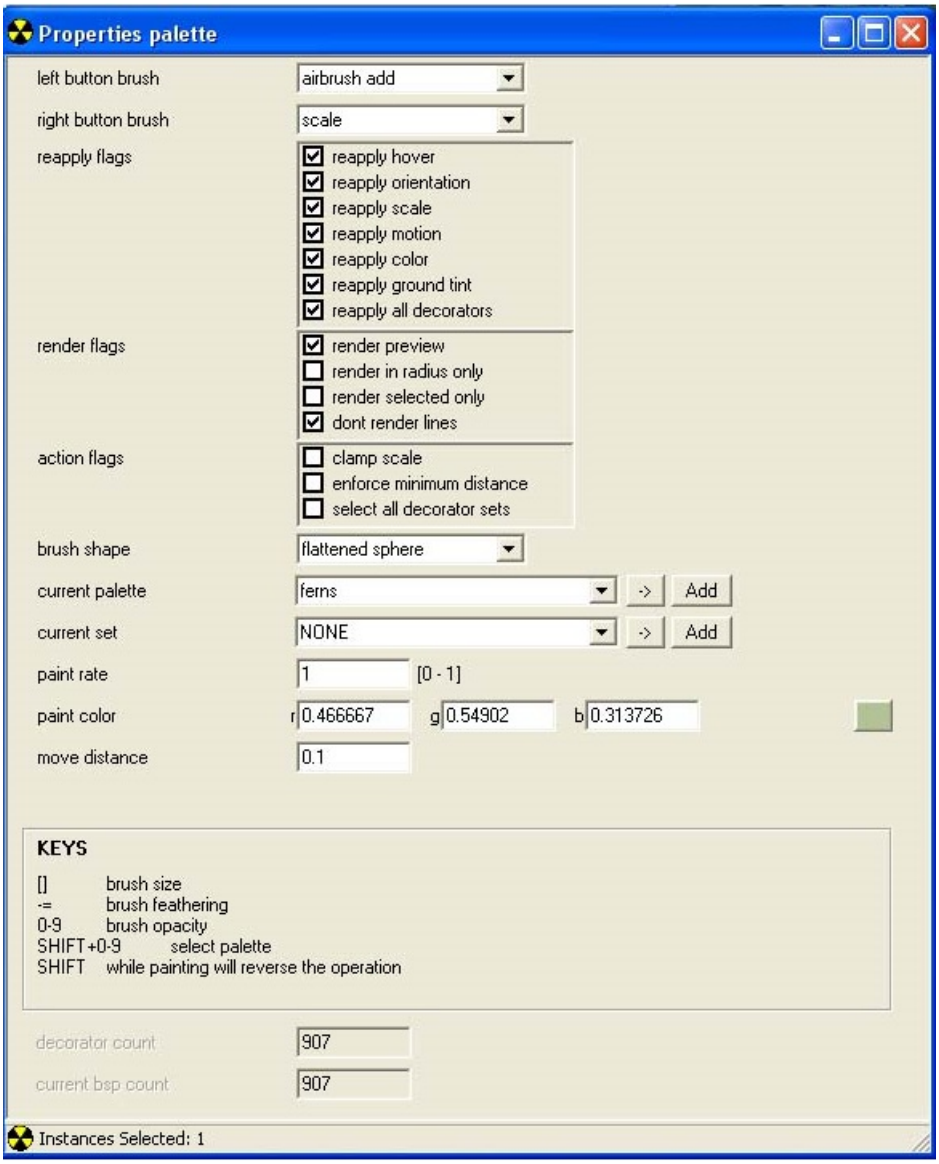


Figure 2 - Decorator Brush Tool.

When you go to paint down the decorators in the game window, two circles should appear and follow your mouse (see Figure 3).

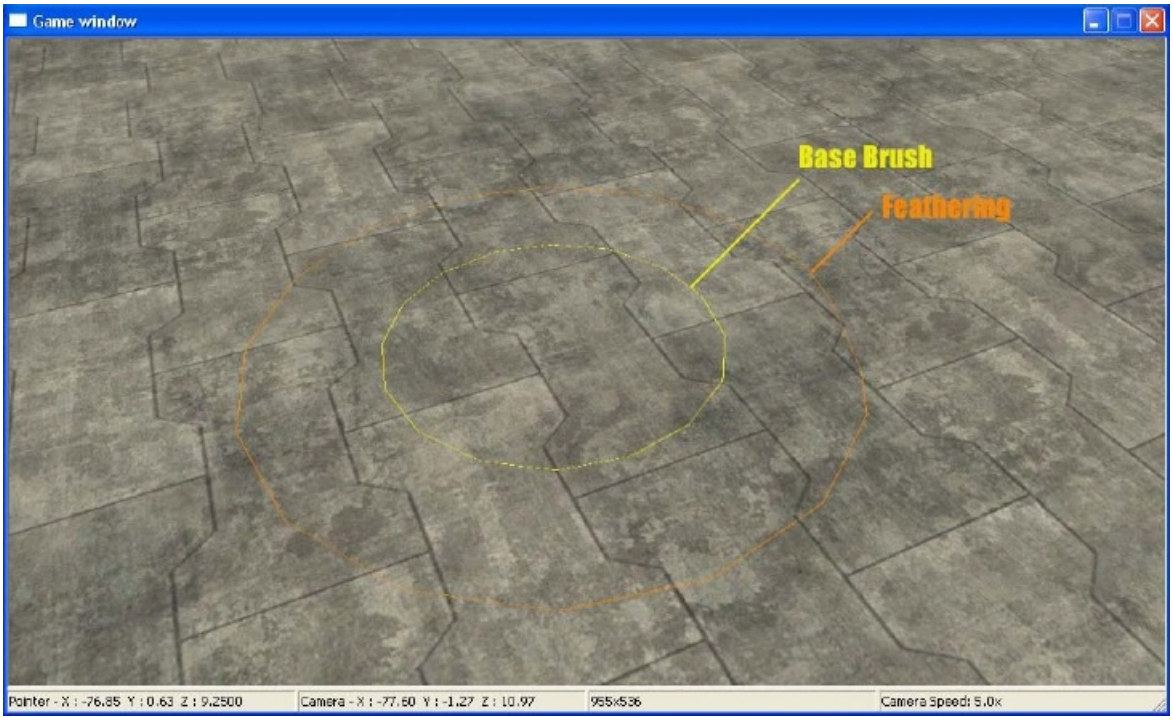


Figure 3 - Brush Example.

The outer orange circle represents the end of the feathering, the inner yellow circle represents the core brush.

Brush key commands

-]— Increase brush size
- [— Decrease brush size
- =— Increase feather size
- -— Decreases feather size
- 0-9— Brush opacity
- shift— Reverse operation (while painting)
- shift+0-9— Select palette

> **NOTE**
> If you see only one circle the brush and feather might be the same size— increase or decrease the size of the feather to see both circles

Tool list and uses

- airbrush add — Allows you to paint down decorators from the current set (or the current palette if the current set is none)
- airburush color— Shifts the color tint for the decorators based upon the color and paint rate, will paint on the current set and current palette
- airbrush erase— Erases the decorators in the current set and current palette and use the paint rate and the brush size/feather
- density smooth— Currently unable to find a function
- precision place— Places a single decorator at the brush center from the current set or, if the current set is set to none, will choose a decorator from the current palette
- precision delete— Deletes a single decorator near to the brush center from the current set or the current palette
- scale — Scales the decorators in current set and current palette to a maximum of 2.0 using the paint rate. Press shift to scale the decorators smaller.

Caution

It's possible to shrink a decorator untill it is no longer visible, so use this tool with care

- scale additive— Does the same thing as scale but the longer it is held down the quicker the scale will happen
- rotate random— Applies a random rotation around the normal of the surface under the decorator to the decorators in the current set and the current palette
- rotate normal— Rotates decorators based on the normal of the surface counter-clockwise (press shift to rotate them clockwise). Rotates both decorators in the current palette and the current set
- rotate local— Rotates decorators counter-clockwise based on their up axis and clockwise when holding

shift. Rotates both decorators in the current palette and the current set

- eraser— Erases the decorators in the current set and current palette if they fall anywhere in the brush boundaries
- reapply type settings — Based upon the reapply flags will "reset" the decorators, see the reapply flags section for more information. This will apply to both the current set and current palette
- drop to ground — Causes the decorators in the current set and palette to drop straight down. This will also use all the reapply flags
- comb— This feature is currently not fully implemented
- thin— Removes decorators that are closer than the minimum distance, affects both the current set and current palette

The following is a list of the other options available and how to use them:

- reapply flags— These are settings used for the reapply type settings and the drop to ground brush tools. If you make any changes to the type settings in a decorator set you will need to use the reapply tool and flag just the areas that you have changed or want changed
 - reapply hover— When checked will reapply the hover setting from the decorator set tag
 - reapply orientation— When checked will reapply the orientation from the decorator set tag
 - reapply scale— When checked will reapply the scale setting from the decorator set tag
 - reapply motion— When checked will reapply the wind/wave settings from the decorator set tag
 - reapply color— When checked will reapply the color settings from the decorator set tag
 - reapply ground tint— When checked will reapply the ground tint settings from the decorator set tag
 - reapply all decorators— When checked will ensure the reapply happens to all decorators in the current set and current palette
- render flags— Flags used to control the rendering of decorators mostly used to improve performance
 - render preview— Renders the actual decorator
 - render in radius only— Renders decorators only in the brush radius (good for very decorator-heavy areas)
 - render selected only— Renders only the decorator types in the current set and current palette
 - dont render lines— Stops the decorator lines from rendering, should always be checked unless debugging or looking for particular decorators
- action flags— Flags that affect the painting of decorators universally
 - clamp scale— Ensures that decorators may only be scaled in the 0.5 to 1.5 range (this function is not yet fully verified)
 - enforce minimum distance— Makes it so that two decorators of the same type cannot be in the minimum radius for that set
 - select all decorator sets— Instead of selecting a current set and current palette will select all decorators
- brush shape— Options to change the shape of the brush

- flattened sphere— The base setting of the brush is a flattened sphere
- spherical— A proper sphere, good for reaching some floating decorators
- tall sphere— A taller sphere, ideal for reaching floating decorators

- current palette — The currently selected palette — see Decorator Palettes for more information about palettes
- current set— The currently selected decorator set
- paint rate— The rate at which the paint brushes operate. If anything is happening too fast you can reduce this number to slow the rate at which the brushes affect decorators.
- paint color— The color used for airbrush color
- move distance

Decorator Palettes

Decorator palettes can be extremely useful for both streamlining performance and editing multiple decorator types at once. To create a decorator palette you will need to select the "Decorator Palette" in the hierarchy view and, like adding decorator sets, you will need to create a new instance to make a palette. Click New Instance (see Figure 4) to create a new instance.

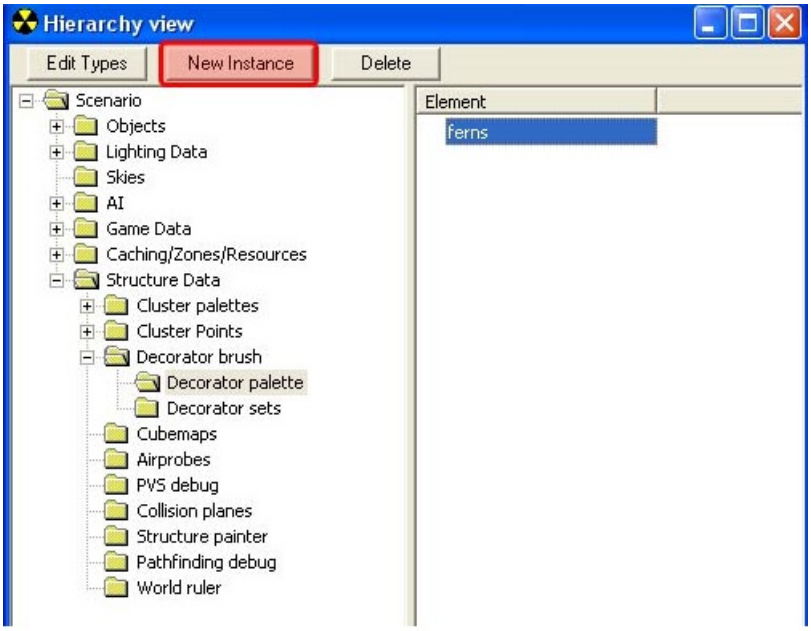


Figure 4 - Decorator Palettes.

When your palette is created you can then add different decorator types to it and weight them to allow you to paint down multiple decorator types at once. It also allows you to filter multiple decorator types to render for a performance boost.

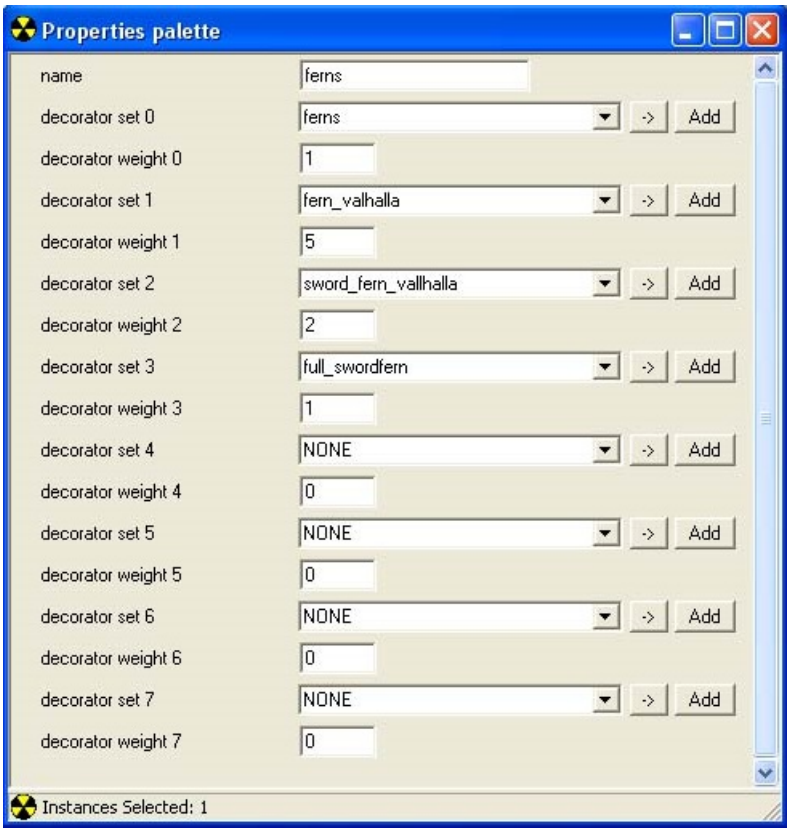


Figure 5 - Decorator Brush Properties.

Tips for painting decorators

- Unlike the rest of Sapien, you can undo and redo when painting decorators. There are multiple levels of undo and they are very helpful when trying to get just the right decorator to appear.
- If you are having performance issues but want to be able to still see multiple decorator sets, use the decorator palette to limit the number of decorator types rendered to just the ones you must see.

Texturing

11/23/2022 • 5 minutes to read

Each decorator set can use one texture, this texture should consist of a 32-bit image (though you should opt for DXT3 compression if at all possible). Decorators are always alpha tested so re-using bitmaps with a specular channel is not possible.

Color Issues With Decorators

Because decorators can be tinted later it's best to try to keep the textures somewhat unsaturated and generally fairly light. This allows more flexibility in the long run. Generally, it works better to use a background slightly darker than the elements that will be visible on the decorator. Because no specular highlight is present it's often good to add a somewhat generic highlight and edge out elements.

Alpha and Decorators

Alpha testing works in a particular manner in decorators and is used when fading out decorators. If you start with anything brighter than 33% then brightness will appear. As you move farther from the decorator in game the percent climbs higher. Because of this it's ideal to create the base map then apply a layer style that feathers the transparency— this also will help with mip related issues.

The feather can be achieved using layer effects on a basic solid white cutout with minimal work (see Figure 1).

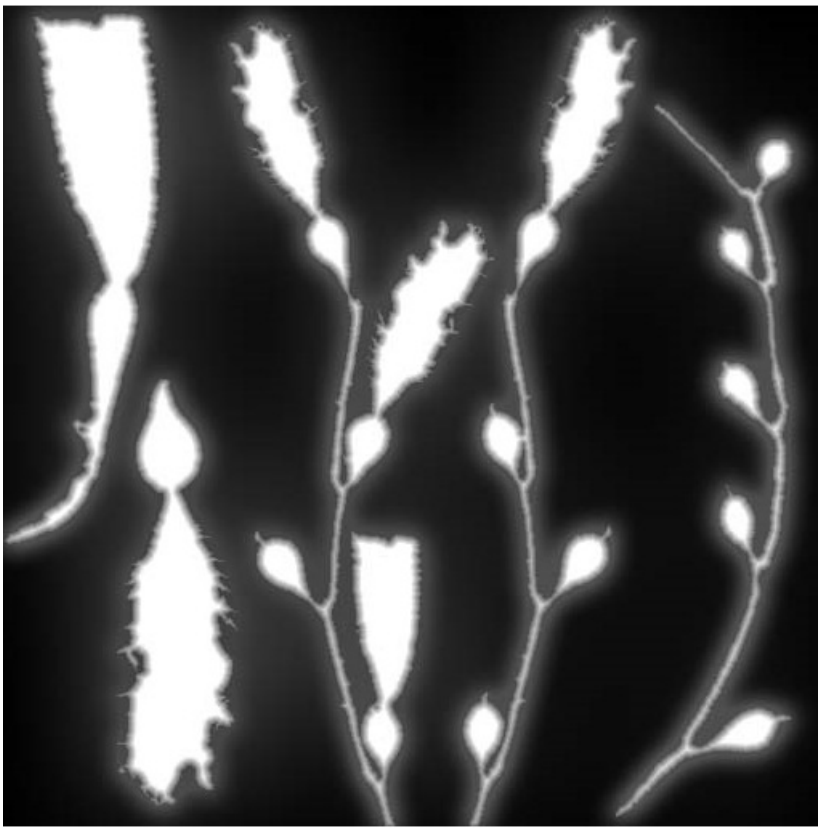


Figure 1 - Alpha and Decorators.

Editing the decorator_set tag

The decorator_set tag defines a group of decorators that can be painted down and contains a link to a render model and a texture. The set itself has a number of settings and also includes several options that can be set for

each mesh in the set.

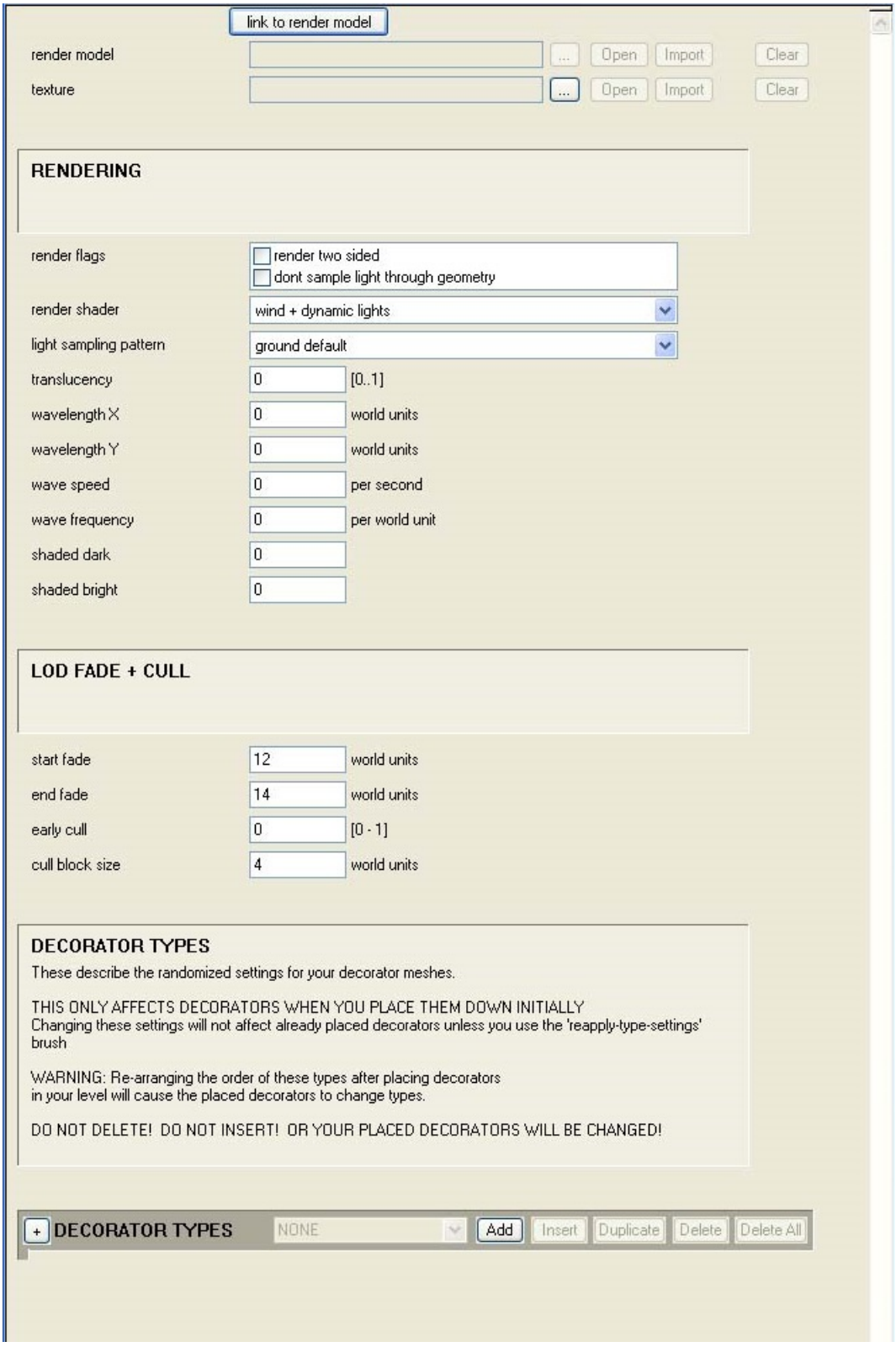


Figure 2 - Decorator_set Tag.

Set Level Settings

Following are set level settings and their functions.

Some settings have descriptions ending with a question mark, which indicates their best-known function. More information about these settings is forthcoming.

- render flags

 - render two sided — Causes both sides of any polygons in the mesh to be rendered
 - dont sample light through geometry — (there is currently no information on this setting)
- render shader
 - wind + dynamic lights — Affected by wind (per BSP) and dynamic lights
 - still + dynamic lights — Only affected by dynamic lights
 - still + no lights — (there is currently no information on this setting)
 - still + sun light only — Still and lit by the sun setting
 - wavy + dynamic lights — Affected by a per-set wave setting (for underwater plants)
 - shaded + dynamic lights — Best lighting option
- light sampling pattern
 - ground default — Samples down from the center of the bounding box
 - hanging — Samples up from the center of the bounding box?
- translucency — Defines the percent of light passed from one side of a card to another (.5 generally gives a good result)
- wavelengthX — Used for the wavy + dynamic lights shader?
- wavelengthY — Used for the wavy + dynamic lights shader?
- wave speed — Used for the wavy + dynamic lights shader?
- wave frequency — Used for the wavy + dynamic lights shader?
- shaded dark — Used for the shaded + dynamic lights shader?
- shaded bright — Used for the shaded + dynamic lights shader?
- start fade — The distance, in world units, to start fading out the decorator
- end fade — The distance, in world units, where the decorator fades completely
- early cull — Starts culling of verts sooner than they would otherwise
- cull block size — The more sparse the decorators the larger this should be

Mesh Level Settings

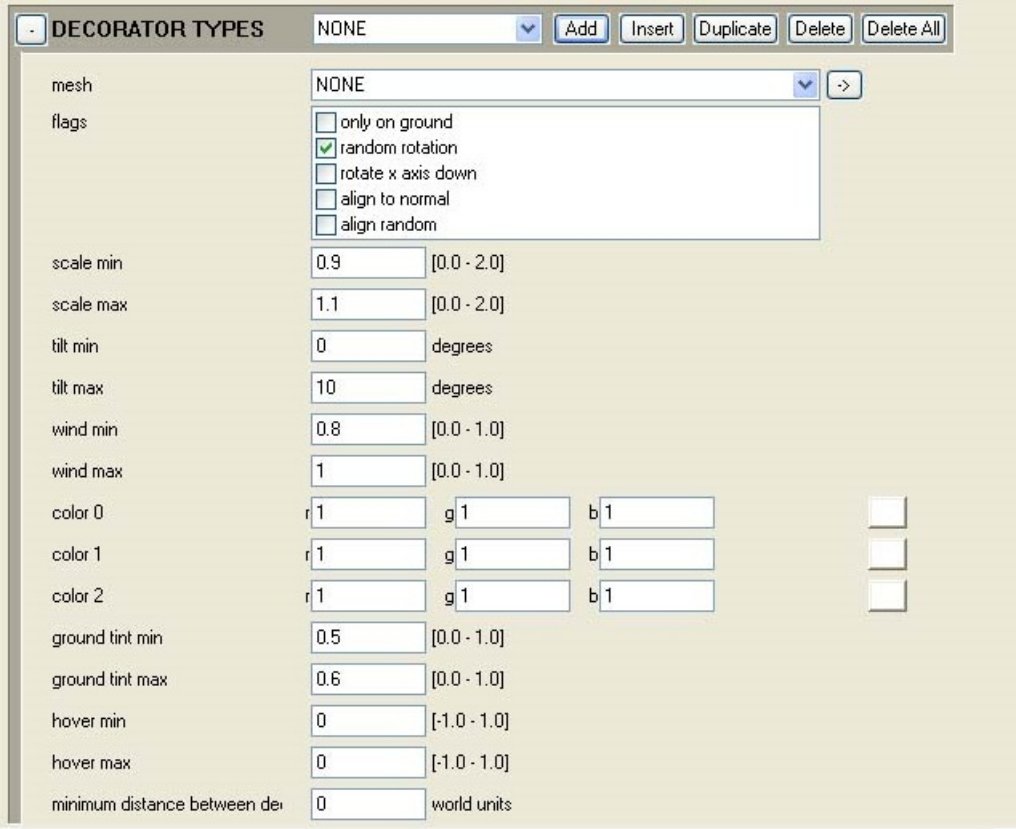


Figure 3 - Mesh Level Settings.

- mesh — This pull down sets which mesh is used for this type

> **NOTE**
> The type takes on the name of the mesh. If you wanted a particular mesh to appear more frequently then you can set more than one type to use that mesh

- flags
 - only on ground — Decorators with this setting will not be placed on surfaces whose angle is greater than ~45 degrees from flat
 - random rotation — randomly rotates the decorator on its Z axis
 - rotate x axis down — rotates the decorator X axis to face toward the ground
 - align to normal — aligns the decorator's Z axis to the face normal of the surface it is painted on
 - align random — randomly aligns the decorator's Z axis
- scale min — Sets the minimum scale value for the decorator

> **NOTE**
> A setting of 0 or a small fraction will cause the decorator to be hard to see

- scale max — Sets the maximum scale factor up to 2.0x the base size. When painted the decorator is scaled to a value somewhere between the min and max
- tilt min — The minimum value in degrees the decorator may be tilted on its X or Y axis
- tilt max — The maximum value in degrees the decorator may be tilted on its X or Y axis

- wind min — The minimum percent that wind affects the decorator (unused for non-wind shaders)
- wind max — the maximum percent that wind affects the decorator (unused for non-wind shaders)
- color0 — The first of three possible colors to tint the decorator when painted down
- color1 — The second of three possible colors to tint the decorator when painted down
- color2 — The third of three possible colors to tin the decorator when painted down
- ground tint min — The minimum amount of tint taken by sampling the diffuse color of the ground under it
- ground tint max — The maximum amount of tint taken by sampling the diffuse color of the ground under it
- hover min — The minimum amount to hover a decorator over a surface
- hover max — The maximum amount to hover a decorator over a surface
- minimum distance between decorators — Does not affect anything

General Notes on the tag

Decorator types are based off of indexes — your painted decorators will work as long as you have the same or a greater number of decorator types. If you re-link your tag to a different render model it may cause phantom meshes to appear. It's possible to remove these by setting Guerilla to expert mode opening the tag and deleting unwanted model instances. This may cause problems. Often the better solution is to just create a clean tag and start over.

Objects

11/23/2022 • 2 minutes to read

Functions

Halo Script Damage_Object

Setting Damage Sections and States

Setting Global Material Types

Model Variants, Regions, Permutations, and States

*Setting Model Variant, Regions, Permutations, and States *

Setting Up Regions and Permutations in 3DS Max

Tagging Damage Sections and States

Halo Script Damage_Object

11/23/2022 • 2 minutes to read

damage_object is a Halo Script command that allows you to apply specific amount of damage to a named object in a scenario.

Use

damage_object is executed from a command line:

damage_object <object name> <material name> <damage amount>

- object name— Named object in scene
- material name— Collision material of named object
- damage amount— Deducted from object's vitality

Example

- damage_object hog hull 100

This command will do 100 damage to the material hull on the warthog named hog on the active map.

Tag Info

There is a hs_damage.damage_effect tag that is linked to the globals.global (see Figure 1).

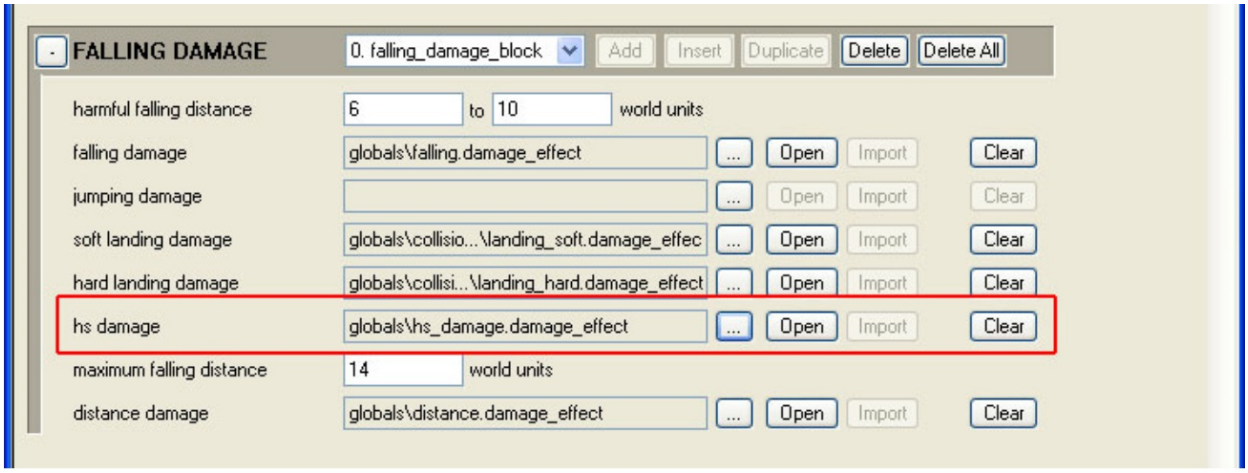


Figure 1 - hs damage tag.

Object Functions

11/23/2022 • 2 minutes to read

There is only one object function that can be read by poops: unique_id, used so that each instance object will have a slightly different value.

Example: You could set a tint range in a poop .shader and then use unique_id to pick a value from that range (see Figure 1 for an example).

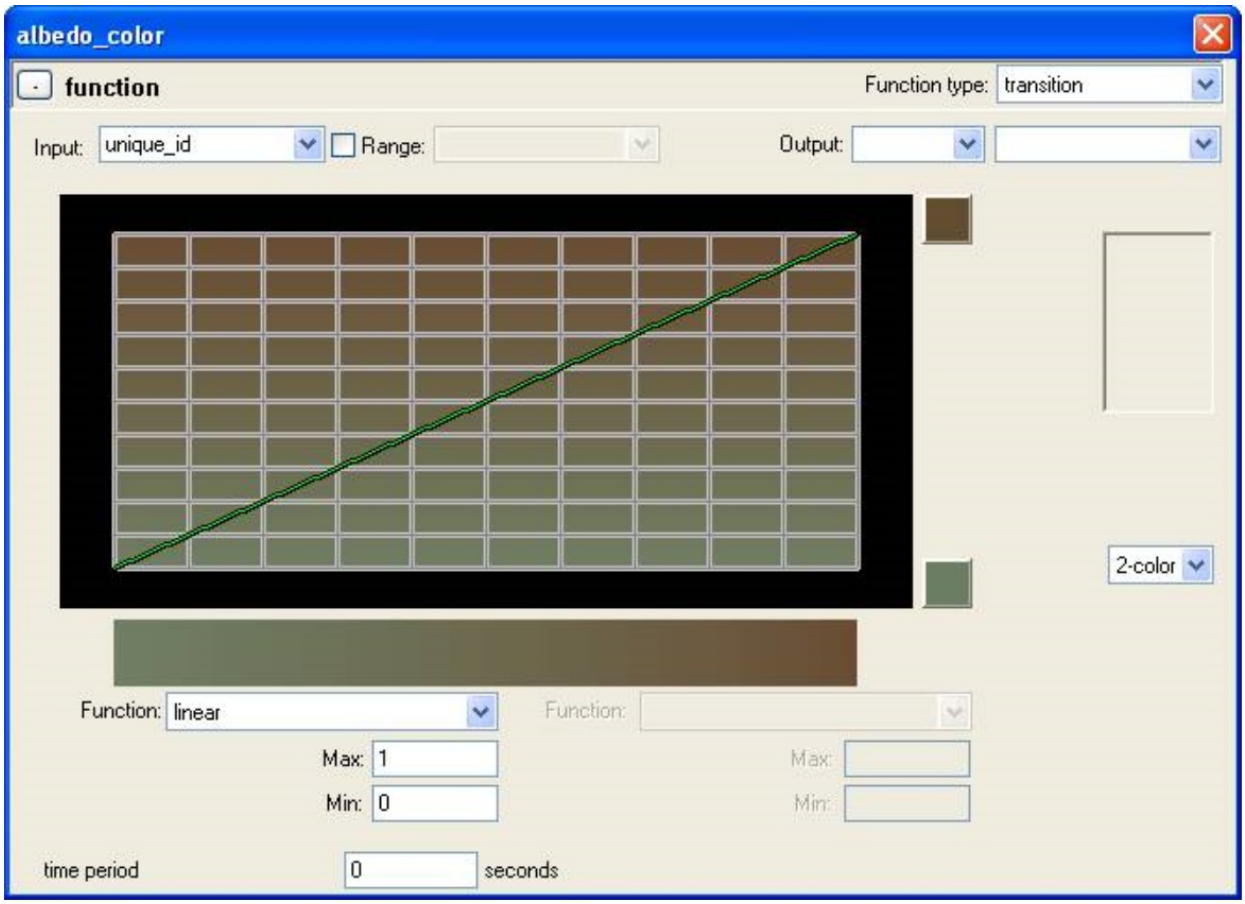


Figure 1 - Albedo Color.

Model Variants, Regions, Permutations, and States

11/23/2022 • 5 minutes to read

This article covers Variants, Regions, and Permutations (for any type of object)

Variants

A **Variant** is a unique form of a specific object (model). For example: We have marines but we also have the ODST and Sergeant Johnson variants of the general marine object. Having separate variants (within the same .model tag) allows you to give an object a unique look while it still shares all of the properties (damage sections, materials, etc.) of the standard version— thus saving you the time and effort of setting up a whole separate object for each type of marine (for example).

The **Variants** block of the .model tag (see Figure 1) is found near the top. It's the first block below the model and LOD information.



Figure 1 - The Variants tag block.

- **name** — The name of the variant you want to create.

> **NOTE**
> If you name the variant default, that variant will always automatically be used by the game engine (unless you specify a different variant in Sapien). Otherwise, the engine will randomly pick variants to use when the object is placed (unless you set a specific variant within Sapien).

Regions

A Region is a homogeneous and somewhat independent section of a model. Regions are set up so that different areas of the same model can have different properties (each region can have different permutations and states than other regions of the model — see below). For example, the head of a marine is one region, while the body is another — both of which have vastly different shapes, colorations, and features. Regions are predefined in 3ds Max before exporting/importing your render/collision/physics models (see Setting regions and permutations in 3ds Max for more information).

The Regions tag block (see Figure 2) is a sub-block of Variants, so if you want to set up regions (and then permutations and states) you first need to have at least one variant set up.



Figure 2 - The Regions tag block.

- **region name**— The name of a region you defined in 3ds Max before exporting the model (you need to enter it manually— this list is not created for you on import).

- **parent variant**— Drop down list which allows you to select the name of the variant you wish this region to inherit all of its properties from.

Permutations

Permutations are different/unique versions of a single region within an object. For example, a Marine object's face is a single region, but we have many different permutations of the face (black, white, male, female, etc). Permutations for each object are created and defined within 3ds Max before the various .JMS files are exported/imported into the engine.



Figure 3 - Three different permutations of the face region on a marine model.

The **Permutations** tag block (see Figure 4) is a sub-block of *Regions*, so if you want to set up any permutations, you must first set up a variant and have at least one region defined.

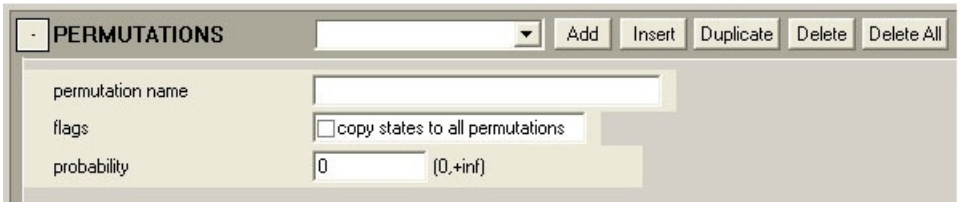


Figure 4 - The Permutations tag block.

- **permutation name**— This is the name of a permutation you want to create for the model. This actually has nothing to do with the permutations you created for the model in 3ds Max— those are used in the damage states (see below).
- **copy states to all permutations** — Takes any states that are assigned to this permutation and copies them to all permutations of this variant of the model. However, it does not display the states when you select any other permutations.
- **probability**— You can set a probability for each permutation to be used when spawned in-game. For example, there are many permutations of marine faces. When a marine is spawned in during gameplay, the engine chooses which face permutation to use based on the probability set in this area of the tag.

States

States are the current condition that any given variant/permutation is in — normally pertaining to a level of damage. Any single permutation can have multiple states. For example, the face of a marine could have a default, minor damage, medium damage, major damage, and destroyed state. These states can be different from the states for any other permutation of a marine face, or they can be same if "Copy states to all permutations" is selected under *Permutations* (flags). You need to set up a separate state for each permutation you set up in 3ds Max of the particular region you are working with.

To set a state for an object, you must first set up a variant, select a region, and set up a permutation.

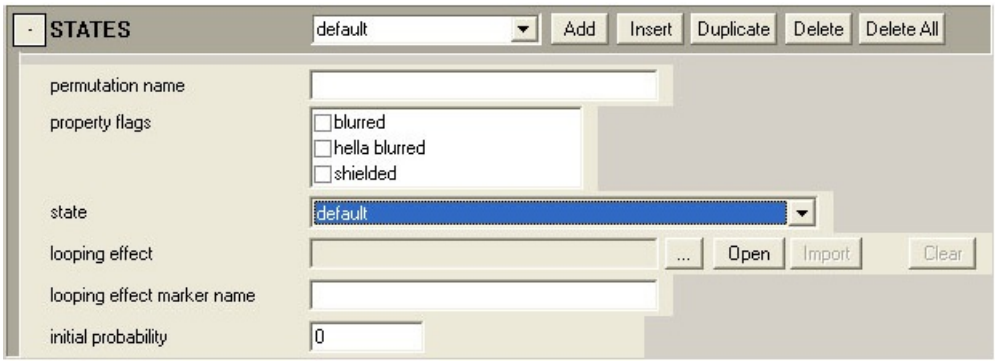


Figure 5 - The States tag block.

- **permutation name**— This is the name of a permutation you specified for a particular region of the model in 3ds Max. If you haven't set up any permutations in 3ds Max, you won't be able to set up any damage states here.
- **property flags**
 - blurred— Visually blurs the permutation in the game. This can be effective when the state is partnered with an effect like smoke or fire, for example
 - hella blurred— More blurring than blurred
 - shielded— Sets the state as shielded
- **state**— A drop down list which allows you to choose from **default, minor damage, medium damage, major damage, or destroyed**. This is the name that will appear above in the heading of the States tag block (or in the drop down list).
- **looping effect**— Links to an effect tag that will coincide with the current state. For example, you could set up a sparking, burning, or smoking effect to go along with a damaged state.
- **looping effect marker name**— The name of the marker you set up in 3ds Max where the effect (looping effect) will emanate from on the model.
- **initial probability**— You can set up a probability for any state to for any permutation to appear initially. For example, if you sometimes wanted a slightly wounded marine to be automatically spawned into the game, you would set up a probability for that state (along with a probability for all of the other states of the marine).

Setting Damage Sections and States

11/23/2022 • 2 minutes to read

To set up working damage sections and states, there are a number of things that need to be configured within the .model tag. They are:

1. In Guerilla, open up the .model tag for your object/model and add an instance of the **New Damage Info** block.
2. Select both a **collision damage reporting type** and a **response damage reporting type** from the drop-down lists (these are within the *New Damage Info* tag block).
3. Define the Vitality/Health properties of the model by Filling in the **body** section of the *New Damage Info* tag block (the shield section is optional— if you want the model to be shielded, fill it in).
4. Add one or more **Damage Sections** to the model (click the Add button in the *Damage Sections* tag block). For the object to be set up correctly, you need to add one damage section per region of the object. The *Damage Sections* tag block is a sub-block of *New Damage Info*.
5. For each Damage Section you add, give it a **Name** and set the **vitality percentage** (this is the sections percentage of the overall vitality of the model/object).
6. Add an **Instant Response** for each damage section you created. The *Instant Response* tag block is a sub-block of *Damage Sections* and won't be available until you've added a damage section.
7. Enter a **Region** of the model where you'd like the response to occur (this is a region you have already pre-defined in 3DS Max).
8. From the drop-down list, select the **state** you'd like the damage section to enter when the instant response fires. In order for the response to play on the object, you must have already set up states for the region of the model you entered in the last step. For more information on setting up Variants, Regions, Permutations, and States, see the article here.
9. Now go back up near the top of the *New Damage Info* block. Select the name of one of your damage sections in the **Indirect damage section** drop down list.
10. Directly above the *New Damage Info* block is the *Materials* block. For each material (one for each collision/physics sub-material you set up in Max), you need to assign a **Damage Section**. Simply pick one of the damage sections you created from the drop down list.
11. Also in the *Materials* tag block, you need to specify a global material name for each material. You can easily select a global material type by right-clicking global material name and then navigating through the sub-menus until you find the material you want.
12. Save

Setting Global Material Types

11/23/2022 • 2 minutes to read

Global Material Types for objects can be set in 3ds Max, but sometimes it's useful to use the collision material name to identify damage sections. If you do, it's important to set the global material type correctly in the tags.

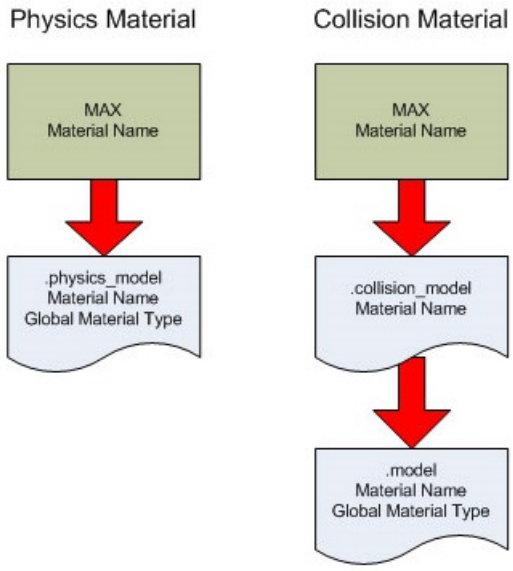


Figure 1 - Physics and Collision Materials.

Setting global material types: collision vs. physics

.physics_model

After import, the .physics_model tag will pick up the material name as set in 3ds Max. You can override this in the .physics_model tag by setting the Global Material Type. Right-click global material name in the tag and a hierarchal menu will appear (see Figure 2).

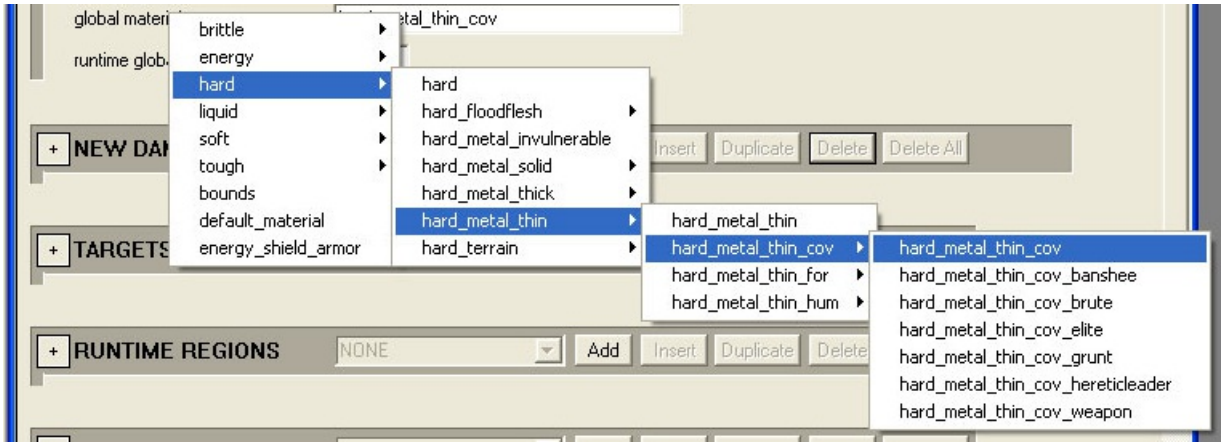


Figure 2 - Setting the global material type in the .model tag.

.collision_model

After import, the .collision_model tag will pick up the material name as set in 3ds Max. **You cannot override this in the .collision_model tag**. The .model tag will pick up the material name from the .collision_model tag. You can override this in the .model tag by setting the Global Material Type.

Setting Model Variants, Regions, Permutations, and States

11/23/2022 • 3 minutes to read

The following are step-by-step setup instructions for configuring a Variant, Region, Permutation, and State for a model.

> **IMPORTANT**
> This section is built from the original documentation and is tailored for use in 3ds Max, but the important principles can be used in any 3d program.

1. Go to the *Variants* tag block (the first block in the tag below the initial information) and add a Variant (click the Add button).
2. Enter a unique name for the variant. If you use the name Default, this will automatically be set as the default variant for the model— meaning that every time you drop one (or one is spawned) in the game, you will get this variant (unless you specify a different variant in Sapien).
3. Under the *Regions* tag block, add a new region (click the Add button). The *Regions* tag block is a sub-block of *Variants* and is located directly below it.
4. For *Region Name*, enter the name of one of the regions you specified for the model in 3ds Max (if you haven't done so, go back and do it, then re-export and import it).
5. For **Parent Variant**, you can select NONE or the name of any variant you have set up. The Region will inherit the properties of whatever variant you select as its parent.
6. Add a new permutation in the *Permutations* tag block (click the Add button). *Permutations* is a sub-block of *Regions* and is located directly below it.
7. For **Permutation Name**, enter any name you would like to call this particular permutation of the model. In many cases, this can simply be default. It has nothing to do with the permutations you already set up in 3ds Max (those are used in the damage states section). The only time you would want to use multiple permutations is if you were going to have different versions of the same object, which could be spawned based on the probability setting in the same tag block (such as the different kinds of marines, for example).
8. If you want every permutation you set up to share the states of the current one, you can check the *Copy states to all permutations box*. Otherwise, leave it empty.
9. Add a new state in the *States* tag block (click the Add button). The *States* block is a sub-block of *Permutations*, which means it's located directly below it.
10. For **Permutation Name**, enter the name of a permutation for the particular region you are working with. You should have specified the permutations for the region in 3ds Max. If you haven't done so, you'll have to go do that, then re-export/import.
11. If you'd like this state to be blurry (perhaps if you're using it in conjunction with some type of smoke effect, for example), check one of the blur checkboxes. Otherwise, you can leave them blank.
12. For **State** select one of the states from the drop-down list. This can be connected later to a damage

section/response. You will need to set up a state for each permutation you created for this particular region in 3ds Max.

13. If you'd like this state of this permutation of this Region of this variant of the model to be connected to some type of effect (smoking, sparking, burning, etc.), you can add one under **Looping Effect**. Click the browse button to find the effects tag you want.

14. If you've selected a looping effect tag, you need to choose a marker on the object to play it from. You need to set these markers up in 3ds Max. Once you've done that (and re-exported/imported), enter the name of the marker in the **Looping Effect Marker Name** text box.

15. Open Sapien. Add your model to the palette, and place one of your objects in the scenario. The default variant will automatically be placed. If you'd like to specify another variant, you can do so by entering the name of the variant in the **Variant** text box of the properties palette.

Setting Up Regions and Permutations in 3DS Max

11/23/2022 • 3 minutes to read

> **IMPORTANT**
> This section is built from the original documentation and is tailored for use in 3ds Max, but the important principles can be used in any 3d program.

This article covers the things you need to know to set up an object with regions and permutations correctly in 3ds Max so that it can have variants and damage sections in the game.

Regions and Permutations

A Region is a homogeneous and somewhat independent section of a model. Regions are set up so that different areas of the same model can have different properties (each region can have different permutations and states than other regions of the model). For example, the head of a marine is one region of the model, while the body is another. Creating regions allows us to have variations between parts of the model (without having to load or create separate models). So, for the face region of the marine, we have many different permutations—male/female, different colors and shapes.

Notes

Region and Permutations are set in the **Name** field of a multi-sub object material in 3ds Max (See Figure 1). The order is "permutation_name" followed by a space, and then "region_name." If it seems counter-intuitive for the sub-part of the hierarchy (the permutation) to be placed before its parent, think of them as being in alphabetical order rather than in hierarchical order.



Figure 1 - Naming Regions and Permutations in the Name field of a multi-sub object material.

- Each permutation of the object can (and probably should) have its own collision and physics models. Each collision section must be manifold (no open edges).
- Collision and Physics permutations should be set up with a separate multi-sub object material than the render materials. This is so that the "Materials" tag block of the .model tag will be populated correctly—with collision material names rather than the names of shaders (see Figure 2).

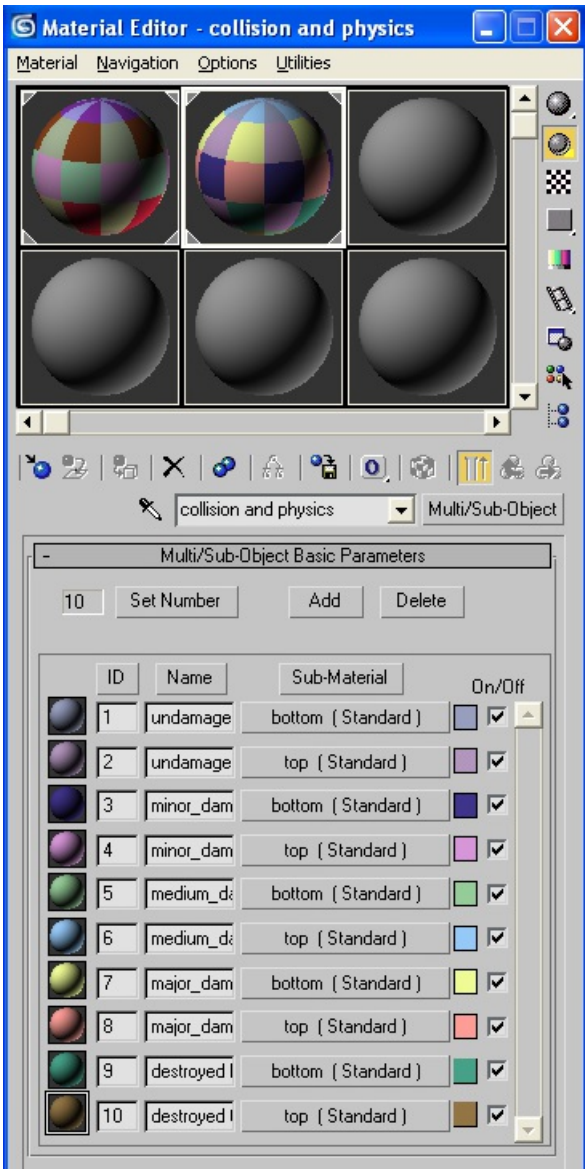


Figure 2 - The Collision and Physics multi-sub object material for a model with two regions (top and bottom) and five permutations.

Step-by-step setup

1. Create the basic render geometry for your object in 3ds Max. Decide how many regions you want the object to have. Remember - for each region, you can have many different permutations. The geometry for each region must be manifold (no open edges), but the regions do not have to be part of the same mesh. For example, if you were making a box with a "top" region and a "bottom" region, you would really just make two separate boxes and place one on top of the other (but not attached to one another).

2. For each region of render geometry that you created in step 1, create two or more permutations. So, you might take region 1, make a copy of it, then move some vertices around (thus creating a different permutation). Then, the original geometry becomes permutation 1 and the new geometry with the moved vertices becomes permutation 2 (both are permutations of region 1).

3. Create a multi/sub-object material in Max. Assign at least one sub-material to each part of your mesh (as you would normally do). However, even if you are planning on using the same shader for different regions or permutations, you will need to assign each one a separate material ID (although they can have the same shader/sub-material name). In the name field of each sub-material, type the permutation name, then the region name for the part of the object that the material is assigned to. For example, if I have the undamaged top half of a box, I would place "undamaged top_half" in the name field (undamaged being the permutation, and top_half being the region). When finished, you should have at least one sub-

material for each region/permutation combination in your model.

4. Now create collision (named with a @) and physics (named with a $) models for each permutation of each region of your geometry. Each collision model needs to be manifold unto itself. Make sure you link all of the geometry (collision, render, and physics) to a frame node (b_) in the max file.
5. Create a new multi/sub-object material for the collision and physics materials. Once again (like you did for the render models) create and assign at least one sub-material for every region/permutation combination. Again, place the permutation and region of each in the Material Name text field. Unlike the render model materials, the collision and physics materials will not become shaders. However, the names of the sub-materials will appear as collision materials in the .model tag later. See Figure 2 for an example.
6. Set up a frame node, bang node, (if you haven't already) and then export and import into guerilla.

Tagging Damage Sections and States

11/23/2022 • 6 minutes to read

This article covers tags for an object with damage sections and states.

New Damage Info Tag Block

You need to have an instance of the global_damage_info_block activated before you can access any of the damage options for your model. It is also important to note that any damage you set up is global to the model. You can't set up a specific damage model for one variant and not have it apply to other variants (for example, all marines share the same damage sections, health, etc).

- **Global indirect material name**: The material type (one of the materials for this specific model - found above in the Materials block) that takes all indirect damage done to the model.
- **Indirect damage section**: The damage section (selected from a drop-down list) that will absorb indirect damage (melee, explosion, etc) for the model.
- **Collision damage reporting type**: For special cases such as the warthog, where the object can do damage that we want attributed to something else. So, when the warthog runs someone over, this is how we attribute the kill to the driver (instead of the hog).
- **Response damage reporting type**: For special cases where the instant response of an object kills a person. For example, when the ghost explodes, or a fusion coil explodes, and kills someone - this property specifies where the damage came from.
- **Body**
 - maximum vitality - The total health (vitality) of the model. For an object like the masterchief, this value would be relatively low (45), because he has low health, but higher shields (maximum shield vitality).
 - minimum stun damage - The minimum amount of damage it takes to stun the object's health. This amount of damage prevents the health from recharging.
 - stun time - The amount of time that the object's health stays stunned before recharging. When this amount of time has expired, the object's health will begin to recharge (unless the minimum stun damage amount is exceeded during that time - at which point the stun timer is reset).
 - recharge time - The amount of time it takes the object to restore itself to full health.
 - recharge fraction - The maximum amount of health that the object will be allowed to recharge. The values go from 0 - 1, so entering .5 would only allow the object to recharge to half of its maximum health.
- **Shield**
 - Shield damaged first person shader - The shader that will be displayed when damage is taken to the shield in first person.
 - Shield damaged shader - The shader that will be displayed when damage is taken to the shield in third person.
 - maximum shield vitality - The total health/strength of the object's shield. For the Master Chief, this number is set to 70.

- global shield material name - The material type for the shield (right-click on the words "global shield material name" to select the material type).
- minimum stun damage - The minimum amount of damage that is required to stun the shield's health (and prevent it from recharging).
- stun time - The amount of time that will elapse before the shield will begin to recharge after being damaged or stunned.
- recharge time - The total amount of time it takes for the shield to recharge.
- shield damaged threshold - The amount of damage it takes to do damage to the shield.
- shield damaged effect - The effect that will play when the shield is damaged.
- shield depleted effect - The effect that will play when the shield is completely depleted (destroyed).
- shield recharging effect - The effect that will play when the shield is in the process of recharging.

Damage Sections

Any area of your model that you wish to show damage needs a damage section set up for it.

- **name**: A unique name given to each damage section
- **flags**: Options you can set for the selected damage section
- **vitality percentage**: This damage section's percentage of the overall health/vitality of the object.

Instant Response Tag Block

The *Instant Response* section of the .model tag configures the response that will take place for each damage section.

- **response type**: A drop-down list with three options: receives all damage, receives area effect damage, and receives local damage . If receives local damage is selected, the selected damage section will not receive area of effect or indirect (melee, grenade, etc) damage.
- **constraint damage type**: A constraint is a piece of the model that holds two pieces together (think of a hinge on a door). If you set up a constraint group (by naming it below under Constraint Group Name, you can use this field to set the type of damage that is done to the constraint - either loosening or destroying. Loosening damage activates a constraint, while destroying damage sets the pieces free (separates them).
- **flags**: Set any of these for special ways the damage section will respond to damage.
- **damage threshold**: The point the damage section's health needs to be at before the instant response will fire. 1 is the object at full health, 0 is the object destroyed.
- **transition effect**: The effect that plays while the object is transitioning from one state to another.
- **transition damage effect**: The effect that plays while the object is transitioning from an undamaged to a damaged state.
- **region**: The region of the object that displays the instant response (the region where the damage section is to be attached). This is a named region that should have been set up in 3DS Max and then defined earlier in the tag. For information on how to set up variants, regions, permutations, and states, check out the article here.
- **new state**: A drop-down list of new states for the region of the object to enter when the instant response fires. This should coincide with a state you have set up for the Region you entered under Region. For

information on how to set up variants, regions, permutations, and states, check out the article here.

- **effect marker name**: This is the name of a marker you should have specified in 3DS Max when creating the model. The marker is the area where the transition effect will render on the model.
- **damage effect marker name**: This is the name of a marker you should have specified in 3DS Max when creating the model. The marker is the area where the damage effect will render on the model.
- Response Delay
 - response delay - The number of seconds until the "instant" response fires.
 - delay effect - The effect that will play during the delay time before the instant response fires.
 - delay effect marker name - The marker (set up in 3DS Max) which specifies where you want the delay effect to render on the model.
- Constraint Destruction
 - constraint/group name - The name of a constraint group that you want this particular instant response/ damage section to belong to. See the entry on Constraint Damage Type above for more info.
- seat ejection
 - ejecting seat label - If the damage section is the seat of a vehicle or turret, you can have the player ejected from it when it is destroyed. Enter the name of the seat that you want to eject the player from here. For example, if you're setting up the damage section for the driver's side of the warthog, you would enter "warthog_d."
- skip fraction
 - skip fraction - If you want this response to fire sometimes, but not always, you can enter a percentage of the time you'd like it to fire. Destroyed Child object marker name
 - Destroyed child object marker name - When the response fires, any child objects created at the supplied marker name (set up in 3DS Max) will be destroyed.
- total damage threshold
 - total damage threshold - The threshold at which damage will be done to the damage section. It defaults to 0, which means any damage done will affect the overall health (vitality) of the damge section.

Instance Objects

11/23/2022 • 4 minutes to read

Instancing is a way of making cheap copies of an object. The math used to render the original can be used again to create the instances. Instance objects are one of the reasons that we switched to the .ASS file format.

> **IMPORTANT**
> This section is built from the original documentation and is tailored for use in 3ds Max, but the important principles can be used in any 3d program.

The following sections cover these topics:

Light Map Resolution

Damage Sections and States Tag Info

Functions

Collision

Pathfinding Policy

Instance Objects in 3ds Max

In order to make poops, a number of criteria must be true of instance objects:

- Be instances in 3ds Max
- Be named with % at the beginning
- Cannot have different modifiers or materials
- Be exported as an ASS file — you cannot copy, reference, add modifiers, or reference different materials in 3ds Max — use instance objects ONLY

And here are some facts worth noting:

- Instances can have uniform scaling — it does not break the instance.
- Non-uniform scaling is simply chucked out at export (when you create the ASS file from within 3ds Max) — it uses the X-axis values for all three axes.
- You can have unique instances in the game— poops that do not share instance definitions with any other objects— they just don't save anything.

Draw Distances on Poops

Putting @30-20 into the suffix of the name of an instance now sets that as its fade pixel sizes (otherwise controlled in the scenario tag for that bsp). It will start fading when its bounding sphere is 30 pixels on screen and not draw at all when less than 20 pixels on screen.

> **NOTE**
> Add letter or character text (like an underscore) after the immediately after the pixel fade numbers in the suffix so extra numbers aren't added by 3ds Max during an auto increment . For example:

%-?name01@10-5_01

The _01 at the end is merely a buffer, making sure the pixel fade distances don't get added to by 3ds Max. Otherwise, instance stencil pixel culling is controlled per-bsp by default instance fading values. In the scenario tag in the structure bsps block at the top. If you leave these at 0 it uses the hard-coded values of 36, 30.

Errors

Instances with open edges will not be collidable.

Markers

You can attach lights and leaf widgets to structure markers. Use debug_structure_markers to see them in-game.

Regions and Permutations

Object Instances can belong to regions and permutations. Naming should be %(permutation region)instance00, just like in the 3ds Max Material Name Field.

This will override the permutation and region set in the 3ds Max material name field.

XRef'ing Instance Objects

Xrefs of instanced objects should now work but nested XRefs (xrefing an object from a file, which itself is an xref from a third file) are not supported well. If you XRef multiple instances, we will write a single object definition but the name of that object may be taken from any one of the instances— not necessarily the 'r;root' object.

We export the name of the xref'ed object in the object definition, not the local object name.

Lighting Nomenclature on Poop Prefixes

Quality of lighting-wise, all three lighting schemes are doing the same per-pixel calculations, so you can get the same look from all three— the only difference is where and how often the lighting is stored.

SUMMARY	-
%	single-probe when your instance is less than 8 world units, and per-pixel when it is greater than 8 world units
%?	per vertex
%!	per-pixel (in the same bitmap as the structure)

Single Probe

Cheap but effective lighting. Footprint memory cost is negligible.

Simply leave the poop without any specific lighting prefix. If your poop has a diameter smaller than 8 world units it will automatically become single probe. Over this size it becomes per-pixel.

```
Example: %probe_01
```

Also, you can force a poop to be single probe. Use the symbol > for this.

```
Example: %>probe_01
```

Single probe poops have prt and are lit like our objects in the world.

Single probe can be effective for cleaning light map errors on single poops (particularly those partially bisecting something else). If your poop has little light transition over the breadth of it— this is a good naming convention to use.

Conversely, single probe poops behave very poorly if the lighting is changing much across the poop. For example, in a lighting gradient, two poops side by side look very bad because there is a harsh lighting line where they meet.

> **NOTE**
> Single probe and per-vertex are lit at the vertex level and can show tessellation artifacts (that is, show the underlying tessellation of the object with lighting variation).

Per Pixel

Per pixel is good to get fidelity and to show lighting variation but it sucks up resolution along with everything else in the lightmap— this is a costly method depending on naming convention used and complexity of poop and shadow contrast thereon.

A prefixing exclamation mark,'!', is used to denote per pixel.

```
Example: %!probe_01
```

This can be augmented with a suffixing number in parentheses. The greater the number the more resolution from the lightmap the object gets.

```
Example: %!probe_01(4)
```

> **NOTE**
> Per-pixel instances do steal resolution from your structure (bsp).

Per Vertex

Uses a separate and additional per-vertex lightmap budget. Cost is dependent purely on complexity/vert count of poop.

A question mark is used to denote per vertex.

```
Example: %?probe_01
```

Per-vertex lightmaps cost us 20 bytes per vertex (this is already compressed). 20 bytes * 1 million verts = 20M. So, a 10 k vert poop would cost 0.2 mbs in per vertex lightmaps (this is in addition to per pixel light maps). Per pixel is good for getting light variation across an object in a specific and cheap fashion (you can cut in where the shadows will fall to eradicate errors/light better).

Collision

11/23/2022 • 2 minutes to read

Environment instance objects (a.k.a. poops) will use the renderable geometry as the collision unless you specify a collision model. It's often advantageous (i.e., cheaper) to use a simpler model for collision.

Setting up Collision in 3ds Max

- The collision model must use the **@** symbol at the begining of the name (it does not need the instance %)
- The collision model needs to be a **child** of the render model
- **Only the original instance needs to have a collision model defined** — it then becomes a resource for the instance and is repeated with every new instance

For example, **%!stair** has the collision model **@stair** associated with it

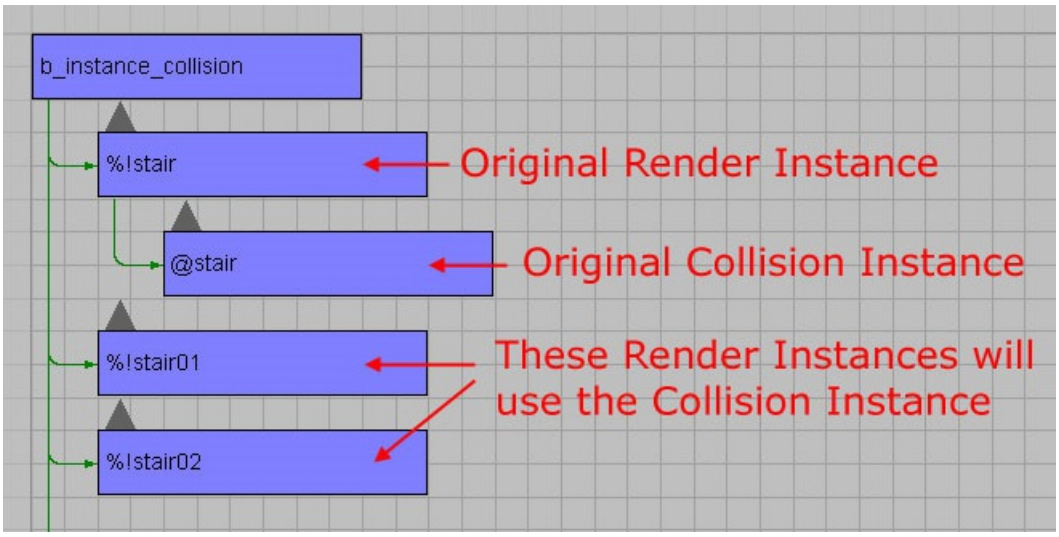


Figure 1 - Hierarchy.

Materials

Collision models use game shaders (either .shader or .shader_terrain) to define their global material type. You cannot input the global material type directly, as you can with object (biped, crate, scenery, etc.) collision.

Lighting Notes

The render geometry is used for lightmapping. Objects sample the render model's lightmap for lighting.

Damage Sections and States Tag Info

11/23/2022 • 6 minutes to read

This article covers tags for an object with damage sections and states.

New Damage Info Tag Block

You need to have an instance of the global_damage_info_block activated before you can access any of the damage options for your model. It is also important to note that any damage you set up is global to the model. You can't set up a specific damage model for one variant and not have it apply to other variants (for example, all marines share the same damage sections, health, etc).

- **Global Indirect Material Name**: The material type that takes all indirect damage done to the model.
- **Indirect Damage Section**: The damage section (selected from a drop-down list) that will absorb indirect damage (melee, explosion, etc) for the model.
- **Collision Damage Reporting Type**: For special cases such as the warthog, where the object can do damage that we want attributed to something else. So, when the warthog runs someone over, this is how we attribute the kill to the driver (instead of the hog).
- **Response Damage Reporting Type**: For special cases where the instant response of an object kills a person. For example, when the ghost explodes, or a fusion coil explodes, and kills someone - this property specifies where the damage came from.
- **Body**
 - maximum vitality - The total health (vitality) of the model. For an object like the masterchief, this value would be relatively low (45), because he has low health, but higher shields (maximum shield vitality).
 - minimum stun damage - The minimum amount of damage it takes to stun the object's health. This amount of damage prevents the health from recharging.
 - stun time - The amount of time that the object's health stays stunned before recharging. When this amount of time has expired, the object's health will begin to recharge (unless the minimum stun damage amount is exceeded during that time - at which point the stun timer is reset).
 - recharge time - The amount of time it takes the object to restore itself to full health.
 - recharge fraction - The maximum amount of health that the object will be allowed to recharge. The values go from 0 - 1, so entering .5 would only allow the object to recharge to half of its maximum health.
- **Shield**
 - Shield damaged first person shader - The shader that will be displayed when damage is taken to the shield in first person.
 - Shield damaged shader - The shader that will be displayed when damage is taken to the shield in third person.
 - maximum shield vitality - The total health/strength of the object's shield. For the Master Chief, this number is set to 70.

- global shield material name - The material type for the shield (right-click on the words "global shield material name" to select the material type).
- minimum stun damage - The minimum amount of damage that is required to stun the shield's health (and prevent it from recharging).
- stun time - The amount of time that will elapse before the shield will begin to recharge after being damaged or stunned.
- recharge time - The total amount of time it takes for the shield to recharge.
- shield damaged threshold - The amount of damage it takes to do damage to the shield.
- shield damaged effect - The effect that will play when the shield is damaged.
- shield depleted effect - The effect that will play when the shield is completely depleted (destroyed).
- shield recharging effect - The effect that will play when the shield is in the process of recharging.

Damage Sections

Any area of your model that you wish to show damage needs a damage section set up for it.

- **name**: A unique name given to each damage section
- **flags**: Options you can set for the selected damage section
- **vitality percentage**: This damage section's percentage of the overall health/vitality of the object.

Instant response Tag Block

The *Instant Response* section of the .model tag configures the response that will take place for each damage section.

- **response type**: A drop-down list with three options: *receives all damage, receives area effect damage, and receives local damage*. If *receives local damage* is selected, the selected damage section will not receive area of effect or indirect (melee, grenade, etc) damage.
- **constraint damage type**: A constraint is a piece of the model that holds two pieces together (think of a hinge on a door). If you set up a constraint group (by naming it below under *Constraint Group Name*, you can use this field to set the type of damage that is done to the constraint - either loosening or destroying. Loosening damage activates a constraint, while destroying damage sets the pieces free (separates them).
- **flags**: Set any of these for special ways the damage section will respond to damage.
- **damage threshold**: The point the damage section's health needs to be at before the instant response will fire. 1 is the object at full health, 0 is the object destroyed.
- **transition effect**: The effect that plays while the object is transitioning from one state to another.
- **transition damage effect**: The effect that plays while the object is transitioning from an undamaged to a damaged state.
- **region**: The region of the object that displays the instant response (the region where the damage section is to be attached). This is a named region that should have been set up in a 3D program and defined earlier in the tag.
- **new state**: A drop-down list of new states for the region of the object to enter when the instant response fires. This should coincide with a state you have set up for the Region you entered under Region.
- **effect marker name**: This is the name of a marker you should have specified in 3DS Max when creating the model. The marker is the area where the transition effect will render on the model.

- **damage effect marker name**: This is the name of a marker you should have specified in 3DS Max when creating the model. The marker is the area where the damage effect will render on the model.
- Response Delay
 - response delay - The number of seconds until the "instant" response fires.
 - delay effect - The effect that will play during the delay time before the instant response fires.
 - delay effect marker name - The marker (set up in a 3D program) which specifies where you want the delay effect to render on the model.
- Constraint Destruction
 - constraint/group name - The name of a constraint group that you want this particular instant response/ damage section to belong to. See the entry on Constraint Damage Type above for more info.
- Seat Ejection
 - ejecting seat label - If the damage section is the seat of a vehicle or turret, you can have the player ejected from it when it is destroyed. Enter the name of the seat that you want to eject the player from here. For example, if you're setting up the damage section for the driver's side of the warthog, you would enter "warthog_d."
- **skip fraction
 - skip fraction - If you want this response to fire sometimes, but not always, you can enter a percentage of the time you'd like it to fire.
- **Destroyed Child object marker name
 - Destroyed child object marker name - When the response fires, any child objects created at the supplied marker name (set up in 3DS Max) will be destroyed.
- **total damage threshold
 - total damage threshold - The threshold at which damage will be done to the damage section. It defaults to 0, which means any damage done will affect the overall health (vitality) of the damage section.

Instance Object Functions

11/23/2022 • 2 minutes to read

There is only one object function that can be read by poops: unique_id, used so that each instance object will have a slightly different value.

Example: You could set a tint range in a poop .shader and then use unique_id to pick a value from that range (see Figure 1 for an example).

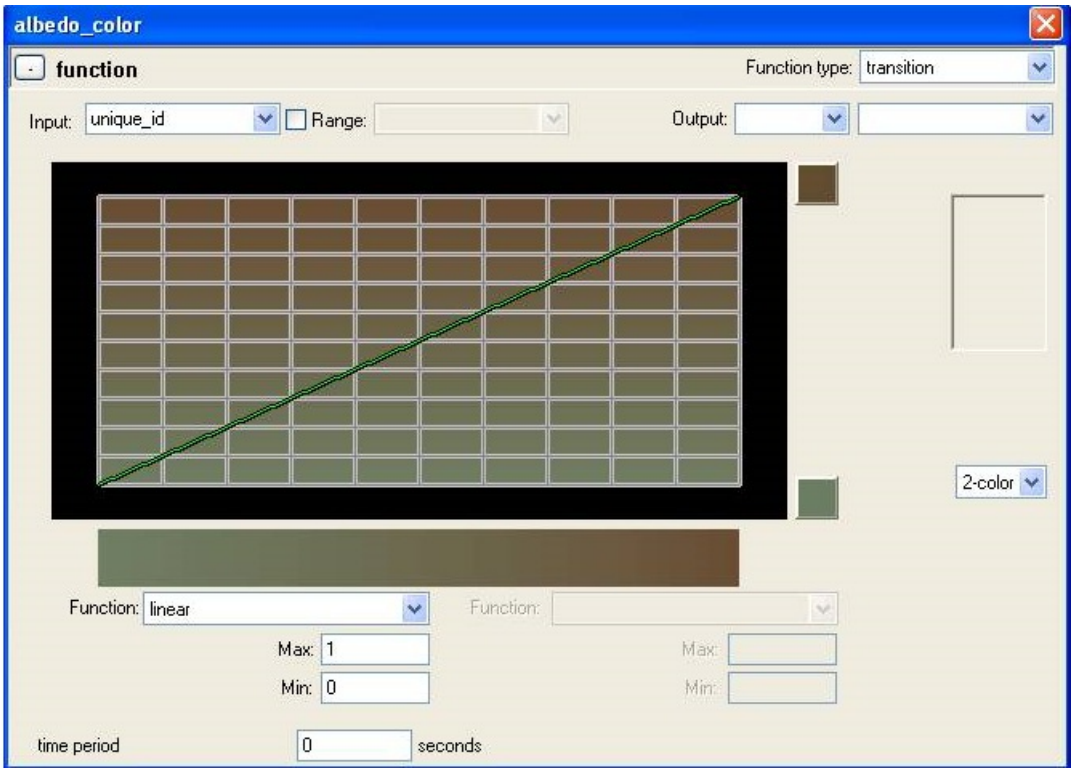


Figure 1 - Albedo Color.

Light Map Resolution

11/23/2022 • 2 minutes to read

When instanced geometry (poops) are initially imported with a BSP, they are given a default lightmap bitmap resolution of 64×64 (the lowest lightmap resolution). We do this because all instances of a definition need to have the same UV coordinates. There are many situations, however, when you may want a piece of instanced geometry that has a lightmap resolution similar to the BSP geometry around it (especially with some larger instances). In those cases, you can set the lightmap resolution specifically for that piece of instanced geometry (and each instance can have a different lightmap resolution, if you want).

The lightmap resolution is set by specifying the multiplier you want to use in the Instance name in 3ds Max. Following the name of the instance, include a number in parentheses. See below for examples.

Lightmap Resolution Settings

SAMPLE NAME OF INSTANCED GEOMETRY	LIGHTMAP BITMAP RESOLUTION
%default_poop	64×64
%mediumres_poop(4)	256×256
%hires_poop(8)	512×512
%superres_poop(16)	1024×1024

> [NOTE] You do not have to set the lightmap resolution as a power of 2 (or as even numbers) as in the examples given here. You can use any number you need in order to make more precise adjustments.

Pathfinding Policy

11/23/2022 • 2 minutes to read

Poops can now have a pathfinding policy specified in their name by adding a special character just after the %.

```
%ex01
```

Poops without any special characters default to pathfinding policy cut-out as brother Damian calls it. This means that AI will pathfind around them but will not be able to walk on them. This is good for things like columns which are z-buffered into the floor. Of the three poop pathfinding policies, this is in the middle in terms of memory usage.

```
%-ex02
```

Poops with a minus just after the percent will use pathfinding policy none. This means that AI will pathfind as if the poop wasn't there. This is good for things that aren't near playable space like rafters on the ceiling. This uses the least amount of memory.

```
%+ex03
```

Poops with a plus just after the percent will use pathfinding policy static. This means that AI will fully pathfind on them. This is good for things like catwalks, stairs, etc. It uses the most amount of memory.

The astute amongst you might be wondering, "Those pathfinding policies are the same as the ones we can specify for scenery and machines, but they have one more called dynamic. What about pathfinding policy dynamic? The answer is that we could add it, but since poops can't move they don't need the dynamic pathfinding policy and it's much more efficient to bake it into the pathfinding graph.

Shaders Overview

11/23/2022 • 2 minutes to read

All shader work is done in Guerilla.

An overview of the different sections of the shader tool follows Figure 1.

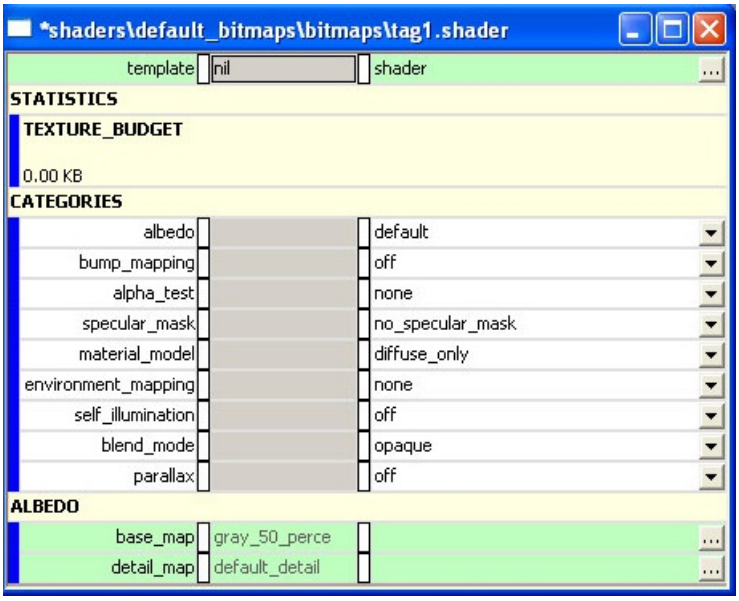


Figure 1 - A sample .shader tag with default settings.

Template

It's no longer necessary to select a specific template for shaders. This is legacy from previous versions.

Statistics

The texture_budget value shows how many kilobytes of memory the current configuration of the shader would use in-game.

Categories

- albedo — (pronounced al-bee-doh) Defines the surface color
- bump_mapping— Defines variations in how lighting affects the surface
- alpha_test— Defines whether a pixel should be rendered or ignored
- specular_mask— Defines how specular properties are controlled on a per-pixel basis
- material_model— Defines what method is used to define specular properties
- environment_mapping— Defines what source is used for the environment map contribution to the specular properties
- self_illumination— Defines which pixels over-ride object lighting
- blend_mode— Defines how an object is added to the framebuffer (how it is blended with the objects behind it)

- parallax— Defines a special kind of bump mapping where pixels may occlude other pixels in the same map

Shaders Articles

Select an article below to learn more about the specific topic.

Albedo Properties

Albedo Blend Properties

Alpha Test Properties

Ambient Coefficient Properties

Ambient Tint Properties

Analytical Anti-Shadow Control Properties

Analytical Specular Contribution Properties

Area Specular Contribution

Blend Mode

Bump Mapping Properties

Diffuse Coefficient

Diffuse Tint

Double Multiply

Environment Map Coefficient

Environment Mapping

Environment Map Specular Contribution

Environment Map Tint

Environment Map Tint Color Properties

Fresnel Coefficient

Fresnel Curve Bias Properties

Fresnel Curve Steepness

Glancing Specular Power

Glancing Specular Tint Properties

Material Model Properties

Material Texture Properties

No Dynamic Lights

Normal Specular Power

Normal Specular Tint

Occlusion Parameter Map

Order 3 Area Specular

Parallax Properties

Rim Coefficient

Rim Maps Transition Ratio

Rim Power

Rim Start

Rim Tint

Roughness Properties

Self-Illumination Properties

Specular Coefficient Properties

Specular Map

Specular Mask Properties

Specular Power

Specular Tint

Standard Properties

Subsurface Coefficient

Subsurface Map

Subsurface Propagation Bias

Subsurface Tint

Templates

Terrain Overview

Transparence Coefficient

Transparence Map Properties

Transparence Normal Bias

Transparence Tint Properties

Albedo Properties

11/23/2022 • 4 minutes to read

Albedo (pronounced al-bee-doh) defines the surface color.

The alpha channel of the base_map (the first bitmap defined in the options below) can be used to control certain other operations in the shader. See specular_mask, material_model, alpha_test and blend_mode.

There are ten different options for albedo parameters.

- default
- detail_blend
- constant_color
- two_change_color
- four_change_color
- three_detail_blend
- two_detail_overlay
- two_detail
- color_mask
- two_detail_black_point

Default

Basic color with one detail map. Plus, you can tint the final color (the combination of the base and detail maps) using the albedo_color value. You can also adjust the final greyscale value of the combined alpha maps using the albedo_color_alpha value.



Figure 1 - Base map with no detail map.

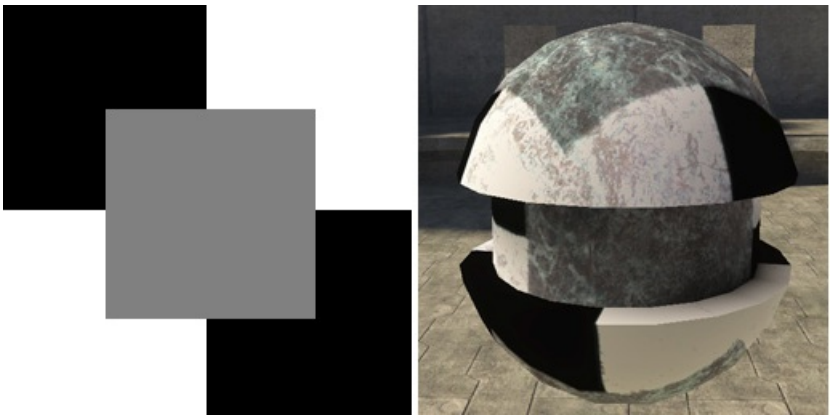


Figure 2 - High contrast detail map and the high contrast detail map applied.

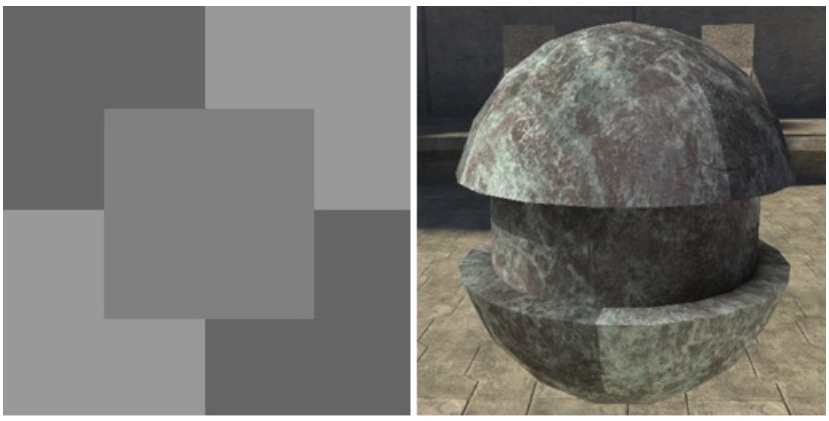


Figure 3 - Low contrast detail map and th elow contrast detail map applioed.

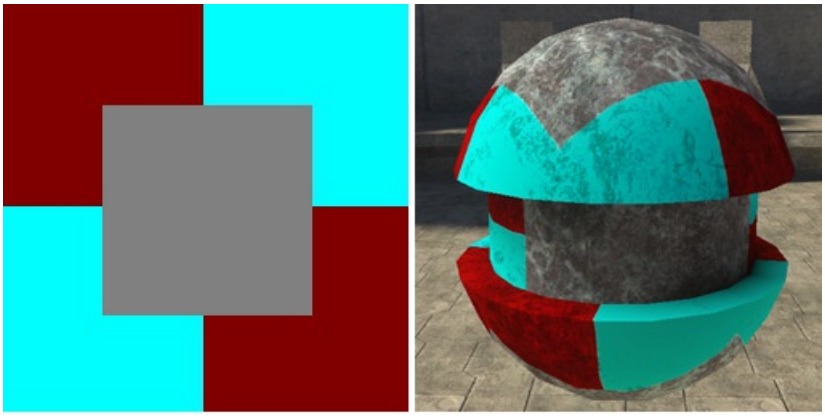


Figure 4 - Color detail map and the color detail map applied.



Figure 5 - Base map with 255 red albedo_color.

- base_map [bitmap] — Initial color map for surface. The lightmapper uses the color of the base_map for

the color of bounced light projected from the surface.

- detail_map [bitmap] — Double multiplies the detail bitmap over the base map. The alpha channel is multiplied with the alpha channel of the base_map for operations requiring alpha information from albedo.
- albedo_color [color value] — Tints the albedo color.
- albedo_color_alpha [value] — Multiplies the grayscale values in the alpha channel by the value you supply. Darken the alpha by multiplying by a number smaller than one, brighten it using a number larger than one.

detail_blend

Detail maps are applied based on the alpha channel of the base map, where black means full detail, and white means no detail. So the alpha channel of the base map acts as a sort of opacity map for the details.

- base_map [bitmap] — Initial color map for surface. The lightmapper uses the color of the base_map for the color of bounced light projected from the surface.
- detail_map [bitmap] — Double multiplies the detail bitmap over the base map, modulated by the alpha channel of the base map. Also, the alpha channel is multiplied with the alpha channel of the base_map for operations requiring alpha information from albedo.
- detail_map2 [bitmap] — Double multiplies the detail bitmap over the base map, modulated by the alpha channel of the base map. Also, the alpha channel is multiplied with the alpha channel of the base_map for operations requiring alpha information from albedo.

constant_color

Constant color with a lovely shade of green

- albedo_color [color] — Initial color for surface.
- albedo_color_alpha [value] — Acts as alpha_channel for operations requiring alpha information from albedo.

two_change_color

- base_map [bitmap] — Initial color map for surface. The lightmapper uses the color of the base_map for the color of bounced light projected from the surface.
- detail_map [bitmap] — Double multiplies the detail bitmap over the base map. Also, the alpha channel is multiplied with the alpha channel of the base_map for operations requiring alpha information from albedo.
- change_color_map [bitmap] — Red and green channels are multiplied with the base map. Change colors are specified in the object tag (.scenery, .crate, .biped, etc.). Red channel is primary change color, green channel is secondary change color.

four_change_color

- base_map [bitmap] — Initial color map for surface. The lightmapper uses the color of the base_map for the color of bounced light projected from the surface.
- detail_map [bitmap] — Double multiplies the detail bitmap over the base map. Also, the alpha channel is multiplied with the alpha channel of the base_map for operations requiring alpha information from albedo.

- change_color_map [bitmap] — Red, green, blue, and alpha channels are multiplied with the base map. Change colors are specified in the object tag (.scenery, .crate, .biped, etc). Red channel is primary change color, green channel is secondary change color, blue is tertiary, alpha is quaternary.

three_detail_blend

- Detail maps are applied based on the alpha channel of the base map, where black means full detail, and white means no detail. So the alpha channel of the base map acts as a sort of opacity map for the details.
- base_map [bitmap] — initial color map for surface. The lightmapper uses the color of the base_map for the color of bounce light projected from the surface.
- detail_map [bitmap] — Double multiplies the detail bitmap over the base map, modulated by the alpha channel of the base map. Also, the alpha channel is multiplied with the alpha channel of the base_map for operations requiring alpha information from albedo.
- detail_map2 [bitmap] — Double multiplies the detail bitmap over the base map, modulated by the alpha channel of the base map. Also, the alpha channel is multiplied with the alpha channel of the base_map for operations requiring alpha information from albedo.
- detail_map3 [bitmap] — Double multiplies the detail bitmap over the base map, modulated by the alpha channel of the base map. Alpha channel is double-multiplied with the alpha channels of the other two detail_maps for operations requiring alpha information from albedo.

two_detail_overlay

- Documentation for this feature is coming soon.

two_detail

Basic color with two detail maps (see Figure 6).

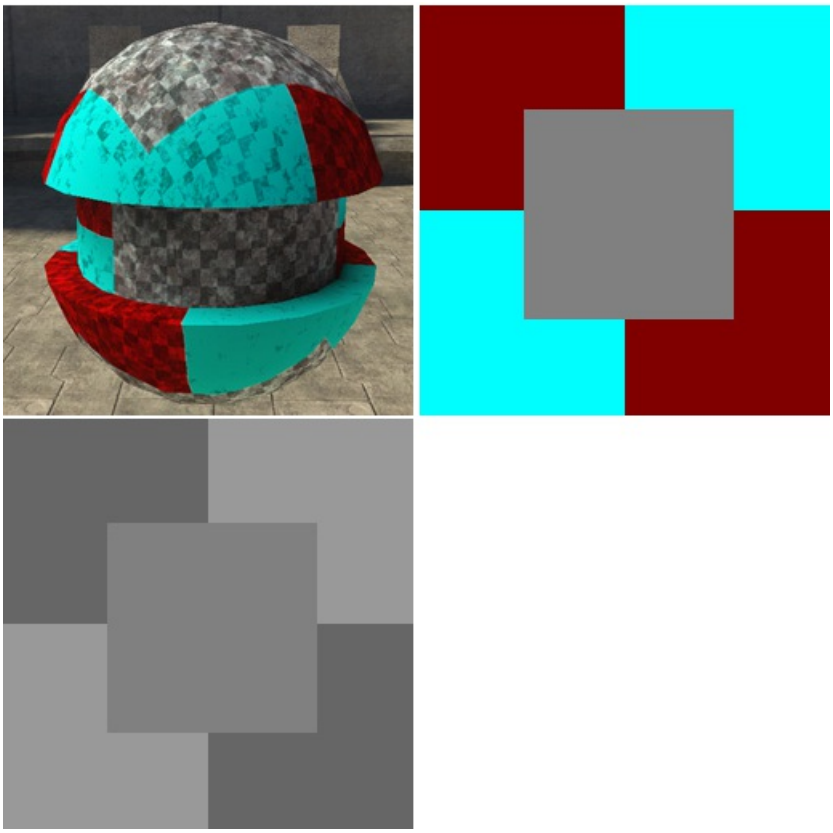


Figure 5 - Base map with two detail maps (left), the color detail map (middle), and low contrast map (right).

- base_map [bitmap] — Initial color map for surface. The lightmapper uses the color of the base_map for the color of bounced light projected from the surface.
- detail_map [bitmap] — Double multiplies the detail bitmap over the base map. The alpha channel is multiplied with the alpha channel of the base_map for operations requiring alpha information from albedo.
- detail_map2 [bitmap] — Double multiplies the detail bitmap over the base and first detail map. The alpha channel is multiplied with the alpha channels of the base_map and detail_map for operations requiring alpha information from albedo.

Albedo Blend Properties

11/23/2022 • 2 minutes to read

Blends between albedo base_map color and the specular tint and fresnel colors defined in the shader.

- 0 = more specular tint (see Figure1).
- 1 = more albedo/less specular tint (see Figure 2).

> **NOTE**
> This can be used as a cheap alternative to a real specular mask.

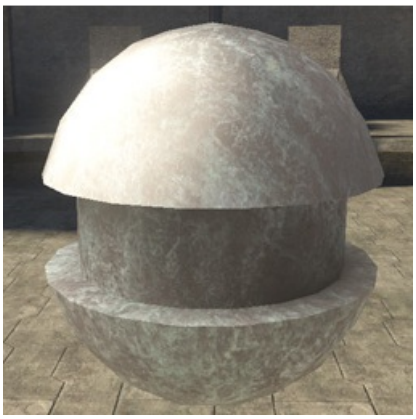


Figure 1 - Albedo 0.



Figure 2 - Albedo 1.

Alpha Test Properties

11/23/2022 • 2 minutes to read

Alpha Testing defines whether a pixel should be rendered or ignored.

> **NOTE**
> This is different than alpha blending. With alpha test, the unused (not rendered) pixels are discarded, whereas they are still used in alpha-blend— thus making alpha test the cheaper solution.

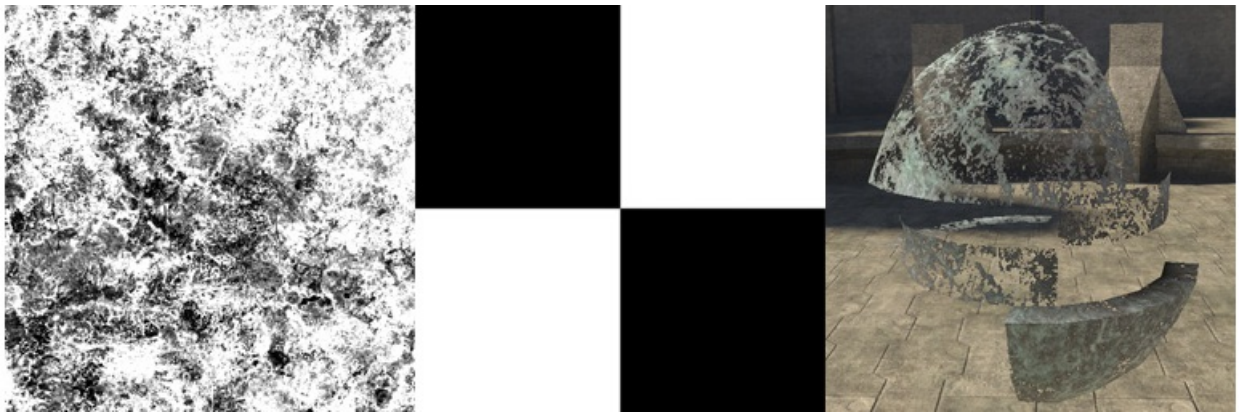


Figure 1 - Two maps combined (left and middle) to affect an object (right).

There are two options for alpha test parameters: None and simple.

None

[no parameters] — No alpha testing. This is the default setting.

Simple

alpha_test_map [bitmap] — Modulates all other maps in the shader based on the color value of the alpha channel. Any pixel with a value less than 128 (50% gray) will not be rendered. Pixels with a value greater than 128 (50% gray) will be rendered on the screen. White in the alpha channel will render, Black will be transparent.

The alpha channels in both the base_map and the alpha_test_map are combined to determine which pixels get drawn and which do not. Grayscale values over 50% are drawn, under 50% are not. The RGB channels of the alpha_test_map are not used.

Ambient Coefficient Properties

11/23/2022 • 2 minutes to read

Adds non-directional light to the model overall, similar to self-illumination but scaled by the overall amount of ambient light in the environment. Good for keeping objects in dark environments from totally disappearing.



Figure 1 - ambient_coefficient: 0.



Figure 2 - ambient_coefficient: 0.5.



Figure 3 - ambient_coefficient: 1.



Figure 4 - ambient_coefficient: 2.

Ambient Tint Properties

11/23/2022 • 2 minutes to read

Tints the ambient lighting (see Figure 1).



Figure 1 - Tinted ambient lighting.

Analytical Anti-Shadow Control Properties

11/23/2022 • 2 minutes to read

This is a way to turn off unwanted analytical specularity in the shadows, which causes shadowy areas to appear brighter than the adjacent lit area.

The valid range is 0-1. 0 is default, meaning specularity is not attenuated at all. 1.0 is pretty aggressive attenuation, so aggressive that it might kill some portion of specularity even in the lit area. You should probably start from 0, and increase the value in small increments to find the sweet spot for your shader. Values in the range of 0.1 to 0.2 generally do the job.

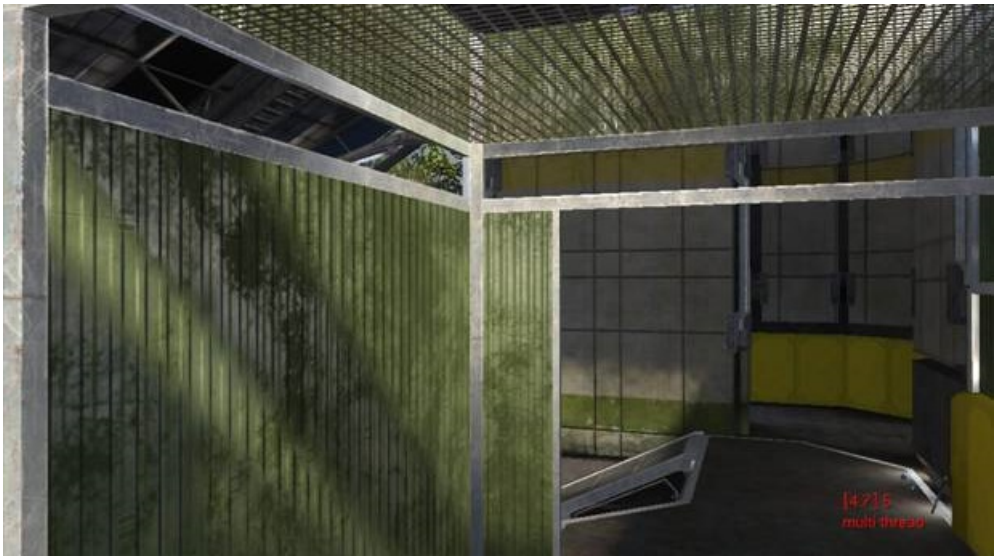


Figure 1 - analytical_anti_shadow_control: 0 (default).

Notice the shadowy area on the left wall is actually brighter than the lit areas.

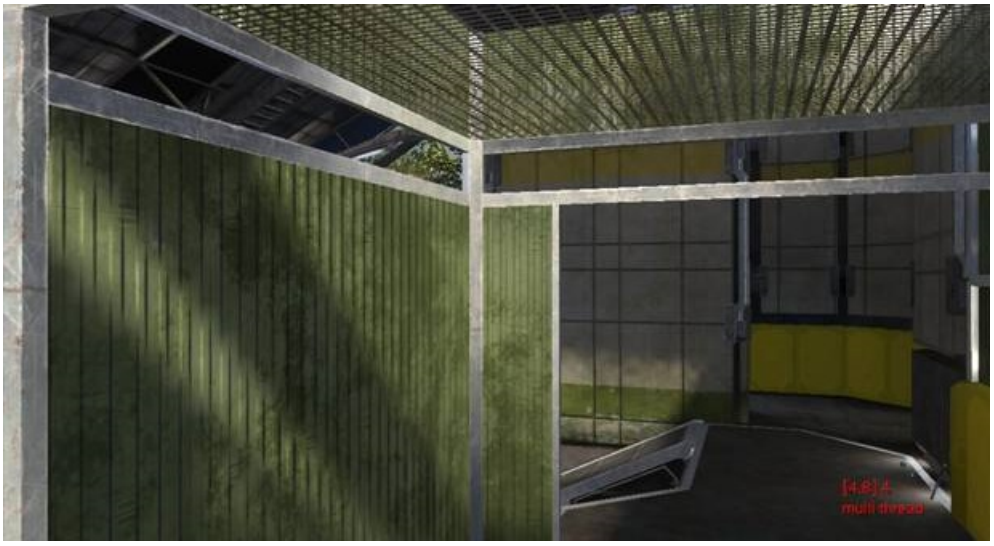


Figure 2 - analytical_anti_shadow_control: 0.15.

Looks much better.

Analytical Specular Contribution Properties

11/23/2022 • 2 minutes to read

What you normally think of as a "specular highlight." Controls the amount of light cast onto the surface from the single brightest light source nearby. Can be focused or spread into different sizes using the roughness parameter.

Defined by a value from 0-1, but higher numbers can be entered for special effects. A value of zero will result in no specular at all.



Figure 1 - analytical_specular_contribution: 0.1.



Figure 2 - analytical_specular_contribution: 0.5.

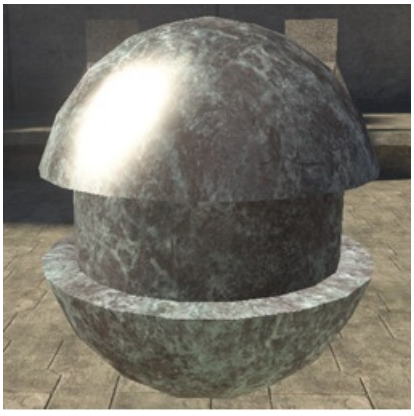


Figure 3 - analytical_specular_contribution: 1.

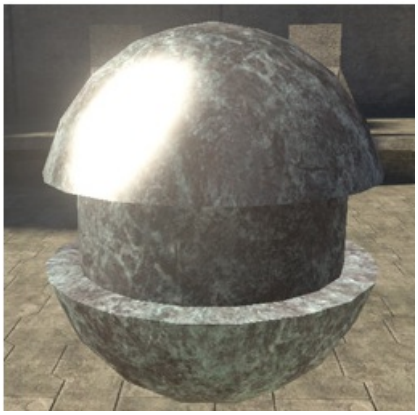


Figure 4 - analytical_specular_contribution: 2.

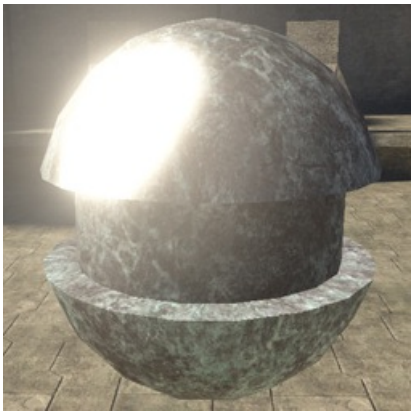


Figure 5 - analytical_specular_contribution: 5.

Area Specular Contribution

11/23/2022 • 2 minutes to read

Ambient lighting. Controls the amount of light cast onto the surface from all light sources in the area. Looks at all of the light probes in the area and calculates the amount and color of light that would be cast on the surface from each. Gives a very accurate representation of area lighting on the suface.

Defined by a value from 0-1, but higher numbers can be entered for special effects. A value of zero will result in no area specular at all. Low numbers work best.



Figure 1 - area_specular_contribution: 0.1.



Figure 2 - area_specular_contribution: 0.5.



Figure 3 - area_specular_contribution: 1.



Figure 4 - area_specular_contribution: 2.



Figure 5 - area_specular_contribution: 5.

Blend Mode

11/23/2022 • 2 minutes to read

Defines how an object is added to the framebuffer (how it is blended with the objects behind it).

> **NOTE**
> The Alpha Blend option uses the alpha channel of the base_map from the albedo.

See the table below for a comparison of how the different blend modes work.

Blend Mode Comparison

BLEND MODE	TRASNPARENT COLOR	WHITE EFFECT	BLACK EFFECT	50% GRAY EFFECT
Additive	Black	Force Framebuffer to White	Transparent	Saturate Framebuffer by 50%
Multiply	White	Transparent	Force Framebuffer to Black	Dim Framebuffer by 50%
Double Multiply	50% Gray	Brighten Framebuffer by 200%	Force Framebuffer to Black	Transparent

> **NOTE**
> Opaque and Alpha_blend blend modes are not included in the table because they are special cases.

There are multiple different blend modes:

- opaque
- additive
- multiply
- alpha_blend
- double_multiply
- pre_multiplied_alpha

opaque

[no parameters]— No transparency. This is the default setting.



Figure 1 - Opaque.

additive

[no parameters]— Reads from the RGB channels in the base_map of the albedo to determine pixel opacity in the scene. Black is completely transparent. As pixel values decrease, the pixels are rendered more transparent. As pixel values increase, they are rendered more opaque.

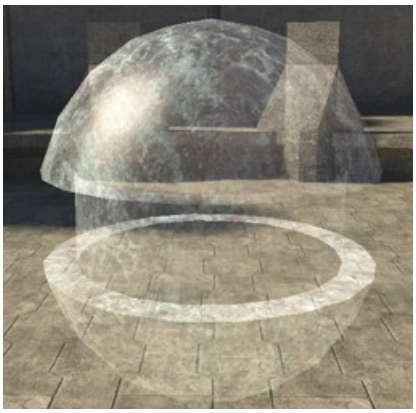


Figure 2 - Additive.

multiply

[no parameters]— Reads from the RGB channels in the base_map of the albedo to determine pixel opacity in the scene. White is completely transparent. As pixel values increase, the pixels are rendered more transparent. As pixel values decrease, they are rendered more opaque.

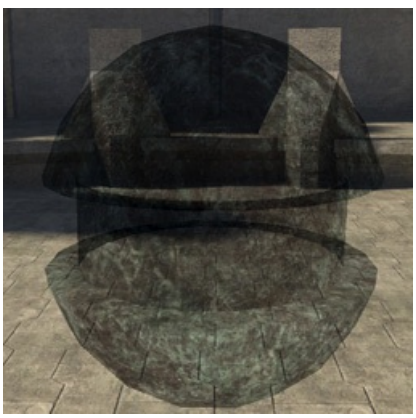


Figure 3 - Multiply.

alpha_blend

[no parameters]— Blends the alpha channel and the RGB channels from the base_map of the albedo to

determine the opacity of the pixels displayed. Darker values (0-128) in the alpha channel of the base_map in the albedo will make pixels more transparent. Lighter values (129-255) will result in more opaque pixel displays.

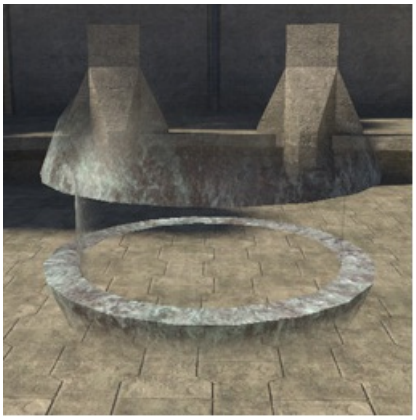


Figure 4 - Alpha Blend.

double_multiply

[no parameters]— Reads from the RGB channels in the base_map of the albedo to determine pixel opacity in the scene. 50% gray is completely transparent. As pixel values increase, they brighten the frame buffer (200% brighter with 255 white) and render more opaque. As pixel values decrease, they darken the frame buffer and render more opaque.

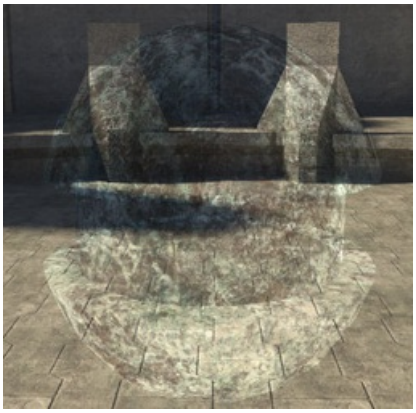


Figure 5 - Double Multiply.

pre_multiplied_alpha

[no parameters]

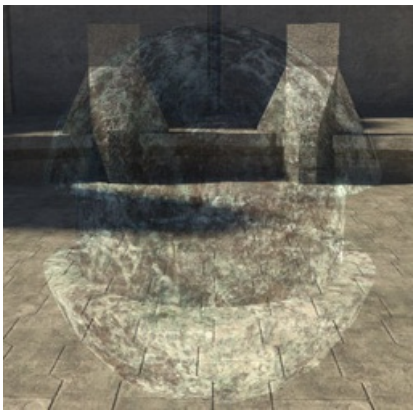


Figure 6 - Pre-Multiplied Alpha.

Bump Mapping Properties

11/23/2022 • 2 minutes to read

Bump Mapping defines variations in how lighting affects the surface.

Gives the illusion of a bumpy surface. This is based entirely on the Value (Lightness/Luminosity/Brightness) of each pixel in the red channel of the bitmap. Each pixel is compared to the one next to it. A large difference creates a deep line, small differences create subtle lines. Specular highlights can also help to accentuate the bump effect.

Note: When authoring bump maps, you should always work in grayscale mode. The strength of the bump effect is controlled by two things: 1) the level of contrast in the bitmap itself, and 2) the bump map height value in the bitmap tag, higher numbers produce more pronounced bumpiness.

See also Shader Parallax Properties.

There are three different options for bump mapping parameters:

off

- **[no parameters]** — No bump mapping. This is the default setting.



Figure 1 - Off.

standard

- **bump_map** — [bitmap] differences in pixel values are used to determine vectors of lighting on the surface. This is based entirely on the Value (Lightness/Luminosity/Brightness) of each pixel in the red channel of the bitmap. Each pixel is compared to the one next to it. A large difference creates a deep line, small differences create subtle lines. When authoring bump maps, you should always work in grayscale mode.



Figure 2 - Standard.

detail

- **bump_map** — [bitmap] differences in pixel values are used to determine vectors of lighting on the surface. This is based entirely on the Value (Lightness/Luminosity/Brightness) of each pixel in the red channel of the bitmap. Each pixel is compared to the one next to it. A large difference creates a deep line, small differences create subtle lines. When authoring bump maps, you should always work in grayscale mode.
- **bump_detail_map** — [bitmap] a second bump_map (height map) for the surface.



Figure 3 - Detail.

Diffuse Coefficient

11/23/2022 • 2 minutes to read

Makes the diffuse color brighter or darker.

Defined by a value from 0-1, but higher numbers can be entered for special effects. A value of zero will render the object completely black.



Figure 1 - Diffuse_coefficient: 0.1.



Figure 2 - Diffuse_coefficient: 0.5.



Figure 3 - Diffuse_coefficient: 1.

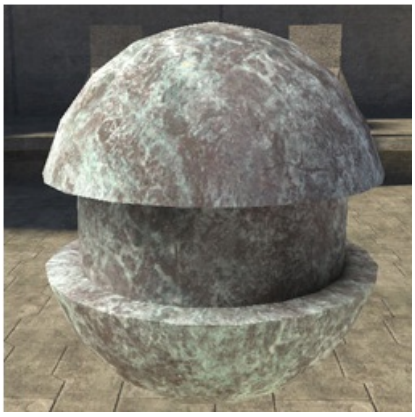


Figure 4 - Diffuse_coefficient: 2.

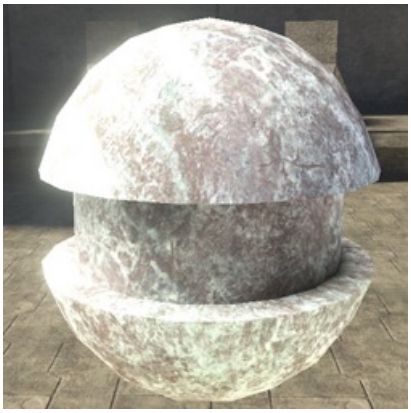


Figure 5 - Diffuse_coefficient: 5.

Diffuse Tint

11/23/2022 • 2 minutes to read

Tints the diffuse color.



Figure 1 - diffuse_tint: default white.



Figure 2 - diffuse_tint: red.

Double Multiply

11/23/2022 • 2 minutes to read

This works the same as the Soft Light layer blending mode in Photoshop, so that a value over 50% gray in the detail map brightens the base map, and values under 50% darken it. A value of exactly 50% gray does nothing.

Environment Map Coefficient

11/23/2022 • 2 minutes to read

A lot like the environment_map_specular_contribution in other material models. Controls the intensity of the environment map.

Environment Mapping

11/23/2022 • 2 minutes to read

Environment Mapping defines what source is used for the environment map contribution to the specular properties.

> **NOTE**
> Any environment map is controlled by the **environment_map_specular_contribution** in the Material Model. You can apply an environment map to your surface without specifying a material model (diffuse only), but you won't have a great deal of control over it.

There are four options for environment mapping parameters:

none

- **[no parameters]**— No environment mapping. This is the default setting.

per_pixel

- **environment_map**— [bitmap] selected bitmap is painted as a cubemap onto the surface.
- **env_tint_color**— [color] The color of light reflected by the environment map.



Figure 1 - per_pixel.

dynamic

Uses the cubemap generated automatically by Tool for the cluster the shader is closest to. To generate cubemaps for a scenario, run **tool cubemaps**. One cubemap is generated per cluster.

- **env_tint_color**— [color] The color of light reflected by the environment map.
- **env_roughness_scale**— [value] Defines the roughness of the surface. A higher value results in light being spread out across the environment map. A small value results in small, pinpoint light on the surface (as if it was a smooth, shiny surface).

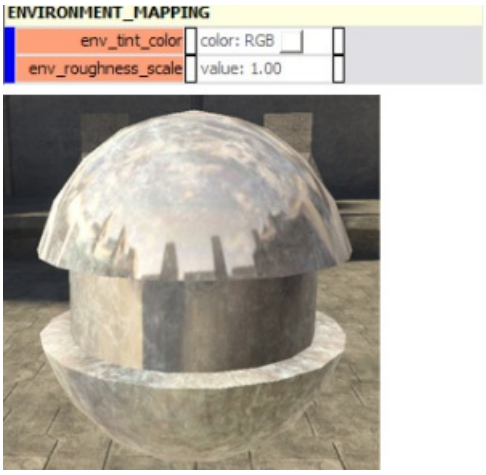


Figure 2 - dynamic.

from_flat_texture

Stretches any flat bitmap around a sphere and uses that as an environment map (the bitmap doesn't need to be set as a cubemap).

- **flat_environment_map**— The bitmap to use as an environment map.
- **env_tint_color**— The color of light reflected by the environment map.
- **hemisphere_percentage**— The percentage of the hemisphere that the texture will be stretched around. Higher values will make the bitmap larger, lower values will shrink the bitmap down.

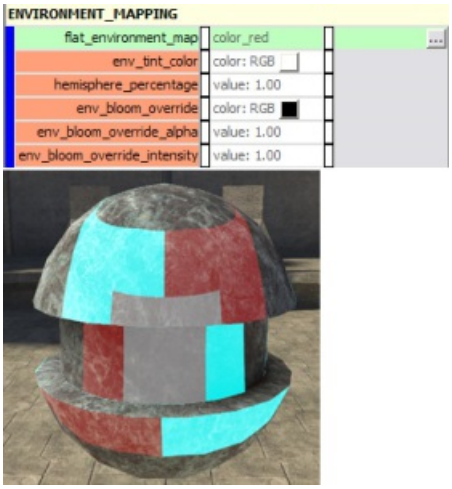


Figure 3 - from_flat_texture.

Environment Specular Contribution

11/23/2022 • 2 minutes to read

Reflection. Set this to 0 if you don't want the environment map to display.

Defined by a value from 0-1, but higher numbers can be entered. A value of zero will result in no reflection at all.



Figure 1 - environment_map_specular_contribution: 0.1.

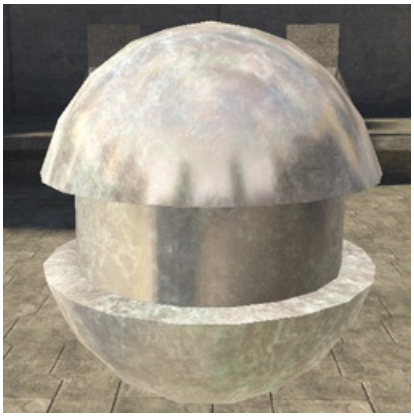


Figure 2 - environment_map_specular_contribution: 0.5.

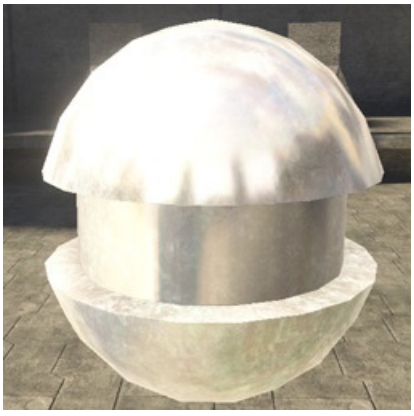


Figure 3 - environment_map_specular_contribution: 1.



Figure 4 - environment_map_specular_contribution: 2.



Figure 5 - environment_map_specular_contribution: 5.

Environment Tint

11/23/2022 • 2 minutes to read

Functions the same as all the other tint parameters. See specular_tint for an example of how it works.

Environment Tint Color Properties

11/23/2022 • 2 minutes to read

Tints the reflection map.

> **NOTE**
>
> Can also be used to adjust the strength of the reflection if you set it to a grayscale color value, black being no reflection and white being full reflection. This lets you set the material_model to diffuse_only and still have control over reflection strength.



Figure 1 - env_tint_color: red.



Figure 2 - env_tint_color: dark gray.

Fresnel Coefficient

11/23/2022 • 2 minutes to read

The "strength" of the environment map. Basically replaces the environment_map_specular_contribution found in other material_model types.



Figure 1 - fresnel_coefficient: .01 (default).



Figure 2 - fresnel_coefficient: 0.25.

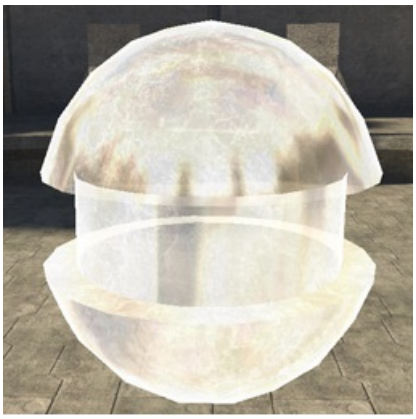


Figure 3 - fresnel_coefficient: 0.5.



Figure 4 - fresnel_coefficient: 1.

Fresnel Curve Bias

11/23/2022 • 2 minutes to read

See Shader fresnel coefficient properties for a description of Shader fresnel curve bias properties.

Fresnel Curve Steepness

11/23/2022 • 2 minutes to read

Controls how far the environment map wraps around from the edges of the model. A value of zero gives you uniform reflection across the model, the same as a non-glass shader. Higher value cause less reflection on the front of the mesh and keep it limited to the sides of the mesh (faces with their normals at 90 degrees to the camera).



Figure 1 - fresnel_curve_steepness: 1.



Figure 2 - fresnel_curve_steepness: 2.5.



Figure 3 - fresnel_curve_steepness: 5 (default).



Figure 4 - fresnel_curve_steepness: 10.

Glancing Specular Power

11/23/2022 • 2 minutes to read

The strength of the specular lighting on mesh faces with normals pointing at 90 degrees to the camera.



Figure 1 - glancing_specular_power: 1.



Figure 2 - glancing_specular_power: 10 (default).

Glancing Specular Tint Properties

11/23/2022 • 2 minutes to read

The color of the specular lighting on mesh faces with normals pointing at 90 degrees to the camera.

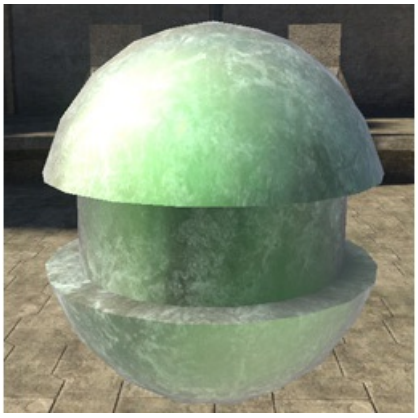


Figure 1 - glancing_specular_tint: white (default).

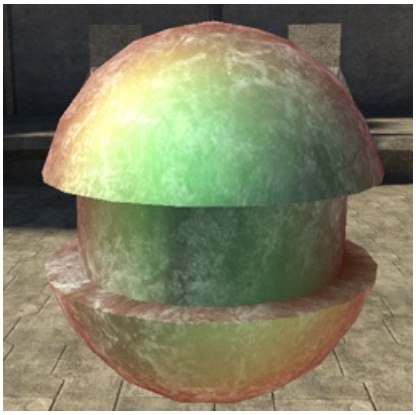


Figure 2 - glancing_specular_tint: [255,0,0].

Material Model Properties

11/23/2022 • 5 minutes to read

Material Model defines what method is used to define specular properties.

> **NOTE**
>
> Per-pixel control of the specular properties is defined in the Specular Mask. The source for the environment mapping is defined in Environment Mapping and can be set to none (which means that there will be no environment mapping, no matter what settings are in the Material Model)

diffuse_only

This gives you diffuse color only, with no specularity.



Figure 1 - diffuse_only.

- **[no parameters]** — No specular properties. This is the default setting.

cook_torrance

A good general purpose shader that gives you per-pixel control over specular, reflection, roughness, and albedo blend.

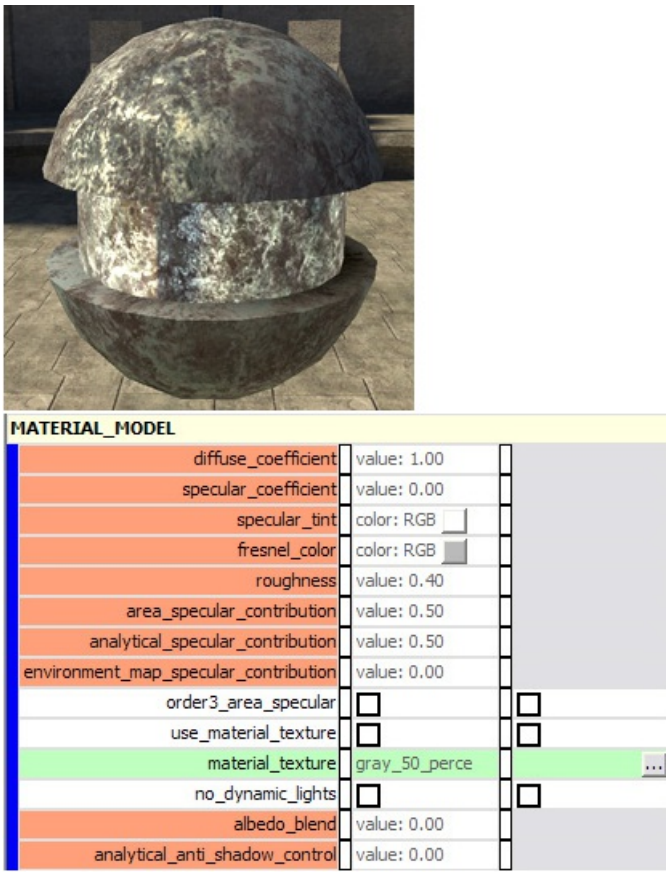


Figure 2 - cook_torrance.

- diffuse_coefficient [value]— Default: 1.0. Controls the amount of albedo base_map rendered on the surface.
- specular_coefficient [value]— Default: 0.0. Controls the overall amount of specular rendered on the surface.
- specular_tint [color]— Default: [255,255,255]. Multiplied with the fresnel_color to control the color of light reflected by the surface. Acts as a "clamp" on the color of light reflected.
- fresnel_color [color]— Default: [128,128,128]. Controls the color of light reflected by the surface in two ways: 1) multiplied by the specular_tint color, and 2) controls the glancing color of light by multiplying all values of the color by 255 (the color of direct, white light).
- roughness [value]— Default: 0.4. Defines the roughness of the surface. A higher value results in light being spread out across the surface. A small value results in small, pinpoint light on the surface (as if it was a smooth, shiny surface).
- area_specular_contribution [value]— Default: 0.5. Controls the amount of light cast onto the surface from all light sources in the area. Looks at all of the light probes in the area and calculates the amount and color of light that would be cast on the surface from each. Gives a very accurate representation of area lighting on the suface.
- analytical_specular_contribution [value]— Default: 0.5. Controls the amount of light cast onto the surface from the single brightest light source nearby. Looks at all of the light probes nearby and picks the single brightest light source. Casts only the light from that single source onto the surface— resulting in pinpoint specs of light.
- environment_map_specular_contribution [value]— Default: 0.0. Controls the environment map's contribution to the specular properties of the shader/surface. Set this to 0 if you don't want the environment map to display.

- order3_area_specular [on/off]— Default: off.
- use_material_texture [on/off]— Default: off. Specify whether or not the material texture bitmap is used.
- material_texture [bitmap] — Various specular properties are controlled by the RGB and Alpha channels of the specified bitmap.
- no_dynamic_lights [on/off]— Default: off.
- albedo_blend [value]— Default: 0.0. Blends between albedo base_map color and the specular tint and fresnel colors defined in the shader. 0 = more specular tint. 1 = more albedo/less specular tint.
- analytical_anti_shadow_control [value]— Default: 0.0. Controls the amount of light cast on the surface to defeat incorrect shadowing (caused by certain combinations of specular properties).

two_lobe_phong

Dual control over specular strength and color. Glancing specular (mesh normals parallel to the screen) can be adjusted seperately from direct specular (perpendicular to the screen).

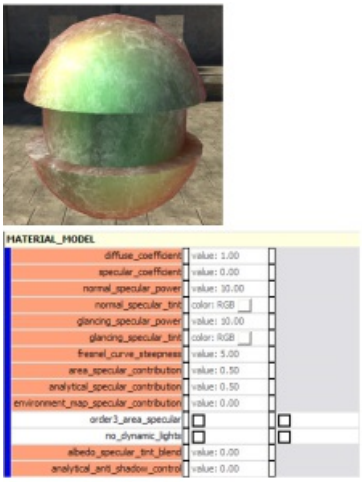


Figure 3 - two_lobe_phong.

- diffuse_coefficient [value]— Default: 1.0. Controls the amount of albedo base_map rendered on the surface.
- ecular_coefficient [value]— Default: 0.0. Controls the overall amount of specular rendered on the surface.
- normal_specular_power [value] Default: 10.0. The amount of colored light that is reflected off of the surface when looking at it from a direct angle.
- normal_specular_tint [color]— Default: [255,255,255]. The color of light reflected off of the surface when viewing it directly.
- glancing_specular_power [value]— Default: 10.0. The amount of colored light that is reflected off of the surface when looking at it from a glancing angle.
- glancing_specular_tint [color]— Default: [255,255,255]. The color of light reflected off of the surface when viewing it from a glancing angle.
- fresnel_curve_steepness [value] Default: 5.0. The steepness of the curve at which normal tint color is turned to glancing tint color. A smaller number makes it easier to see the glancing color (sets glancing to a very wide range of angles). A larger number makes it easier to see the normal color (sets glancing to a very small— narrow— range of angles).
- area_specular_contribution [value]— Default: 0.5. Controls the amount of light cast onto the surface from

all light sources in the area. Looks at all of the light probes in the area and calculates the amount and color of light that would be cast on the surface from each. Gives a very accurate representation of area lighting on the surface.

- analytical_specular_contribution [value]— Default: 0.5. Controls the amount of light cast onto the surface from the single brightest light source nearby. Looks at all of the light probes nearby and picks the single brightest light source. Casts only the light from that single source onto the surface— resulting in pinpoint specs of light.
- environment_map_specular_contribution [value]— Default: 0.0. Controls the environment map's contribution to the specular properties of the shader/surface. Set this to 0 if you don't want the environment map to display.
- order3_area_specular [on/off]— Default: off.
- no_dynamic_lights [on/off]— Default: off.
- albedo_specular_tint_blend [value]— Default: 0.0. Blends between albedo base_map color and the specular tint and fresnel colors defined in the shader. 0 = more specular tint. 1 = more albedo/less specular tint.
- analytical_anti_shadow_control [value]— Default: 0.0.

foliage

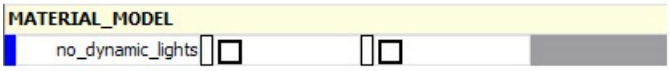


Figure 4 - foliage.

- no_dynamic_lights [on\off]— Default: off.

glass

Mimics the behavior of glass where reflection is more pronounced on the sides of an object, and less so in the center where the faces are more parallel to the camera (also know as a Fresnel effect). Click on the fresnel links for more info. Also, this is specifically meant to be used with some kind of environment mapping. If there is no env map, the sides of the model will be black.

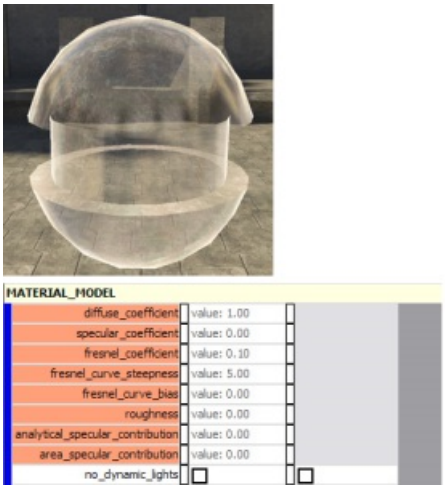


Figure 5 - glass.

- diffuse_coefficient [value]— Default: 1.0.
- specular_coefficient [value]— Default: 0.0.

- fresnel_coefficient [value]— Default: 0.1.
- fresnel_curve_steepness [value]— Default: 5.0.
- fresnel_curve_bias [value]— Default: 0.0.
- roughness [value]— Default: 0.0.
- analytical_specular_contribution [value]— Default: 0.0.
- area_specular_contribution [value]— Default: 0.0.
- no_dynamic_lights [on\off]— Default: off.

organism

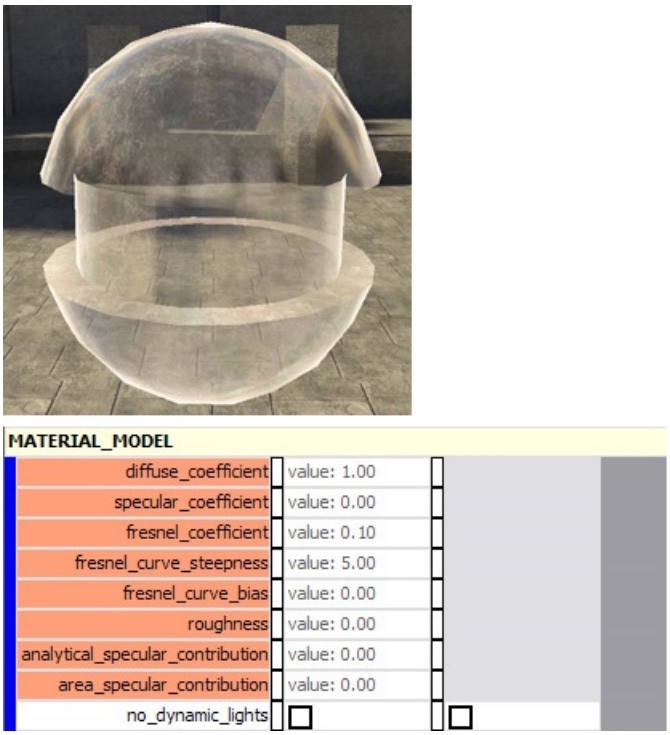


Figure 6 - organism.

- diffuse_coefficient [value]— Default: 0.0.
- diffuse_tint [color]— Default: [255,255,255]
- analytical_specular_contribution [value]— Default: 0.0.
- area_specular_contribution [value]— Default: 0.0.
- specular_tint [color]— Default: [255,255,255].
- specular_power [value]— Default: 10.0.
- specular_map [bitmap].
- environment_map_coefficient [value]— Default: 0.0.
- environment_map_tint [color]— Default: [255,255,255].
- fresnel_curve_steepness [value]— Default: 5.0.
- rim_coefficient [value]— Default: 1.0.
- rim_tint [color]— Default: [0,0,0].

- rim_power [value]— Default: 2.0.
- rim_start [value]— Default: 0.7.
- rim_maps_transition_ratio [value]— Default: 0.0.
- ambient_coefficient [value]— Default: 0.0.
- ambient_tint [color]— Default: [255,255,255].
- occlusion_parameter_map [bitmap].
- subsurface_coefficient [value]— Default: 0.0.
- subsurface_tint [color]— Default: [0,0,0].
- subsurface_propagation_bias [value]— Default: 0.0.
- subsurface_normal_detail [value]— Default: 0.0.
- subsurface_map [bitmap].
- transparence_coefficient [value]— Default: 0.0.
- transparence_tint [color]— Default: [0,0,0].
- transparence_normal_bias [value]— Default: 0.0.
- transparence_normal_detail [value]— Default: 0.0.
- transparence_map [bitmap].
- final_tint [color]— Default: [0,0,0].
- no_dynamic_lights [on/off]— Default: off.

single_lobe_phong

Simple material model with diffuse and specularity.

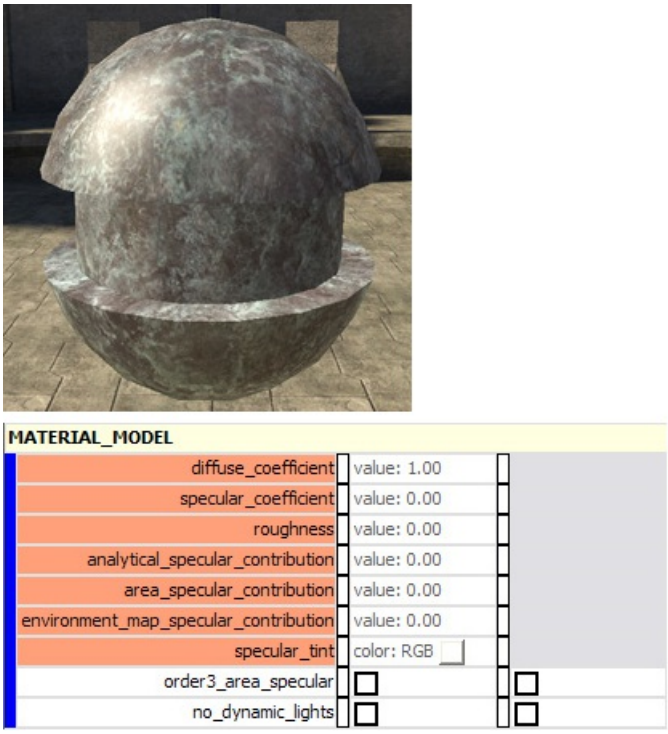


Figure 7 - single_lobe_phong.

- diffuse_coefficient [value]— Default: 1.0.
- specular_coefficient [value]— Default: 0.0.
- roughness [value] — Default: 0.0.

> **NOTE**
> The default of zero doesn't give you any analytical specular, so set it to at least 0.3.

- analytical_specular_contribution [value]— Default: 0.0.
- area_specular_contribution [value]— Default: 0.0.
- environment_map_specular_contribution [value]— Default: 0.0.
- specular_tint [color]— Default: [255,255,255].
- order3_area_specular [on/off]— Default: off.
- no_dynamic_lights [on/off]— Default: off.

Material Texture Properties

11/23/2022 • 2 minutes to read

Available only in the cook_torrance material_model. Allows you to control specular coefficient, albedo blend, environment specular contribution and roughness on a per-pixel basis using the RGB and Alpha channels of the material texture bitmap.

You must check the use_material_texture box to activate the use of the material_texture bitmap. Checking this box disables some of the numeric value inputs.

> **TIP**
> If you decide to use a material texture, you should remove the old specular alpha mask from your diffuse map— this will save you some texture memory.

Red

Specular Coefficient (does not override the number value input).

You can think of this as the same thing as a specular mask, which is why you should remove it from the diffuse map.



Figure 1 - Diffuse map copied into the red channel.

- Black = no specular
- White = max specular as defined by the specular_coefficient and related parameters (area and analytical, but not environment_map, contributions).

Green

Albedo Blend (overrides the number value input).



Figure 2 - Half black and half white.

- Black = specular uses tint color
- White = specular uses albedo color (tinted by the tint color)

Blue

Environment Specular Contribution (overrides the number value input).



Figure 3 - Half black and half white.

- Black = no environment.
- White = max environment.

Alpha

Roughness (overrides the number value input).

- Black = sharp.
- White = wide.

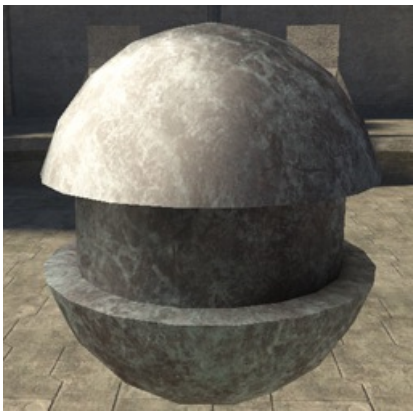


Figure 4 - Uniform flat grey (medium roughness).

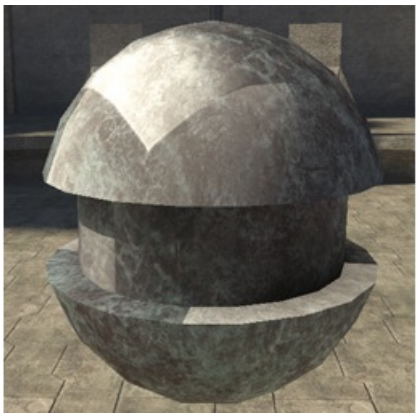


Figure 5 - Checker Map.

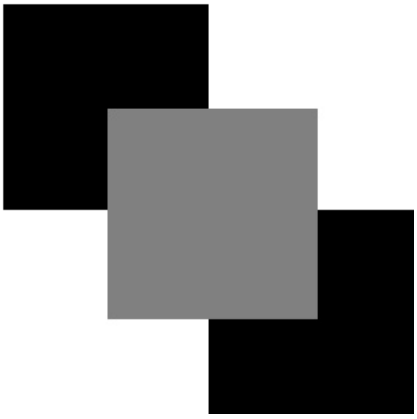


Figure 6 - Alpha Map used in figure 5.

> **TIP**
>
> A good place to start is to copy and invert the specular (red) channel. This gives you tight highlights in the areas with the strongest spec values, which is how spec usually works.

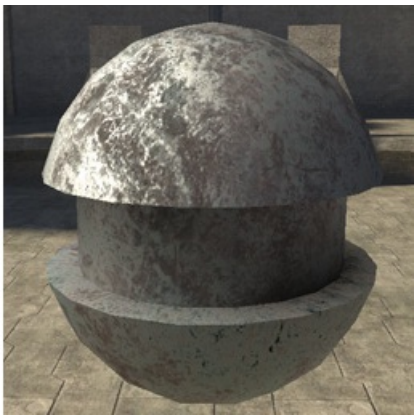


Figure 7 - Spec map with inverted roughness map.

No Dynamic Lights

11/23/2022 • 2 minutes to read

Defines whether or not the shader reacts to dynamic lights.

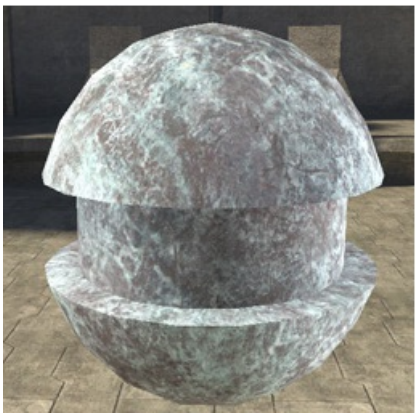


Figure 1 - no_dynamic_lights: off



Figure 2 - no_dynamic_lights: on

Normal Specular Power

11/23/2022 • 2 minutes to read

The strength of the specular lighting on mesh faces with normals pointing at the camera.

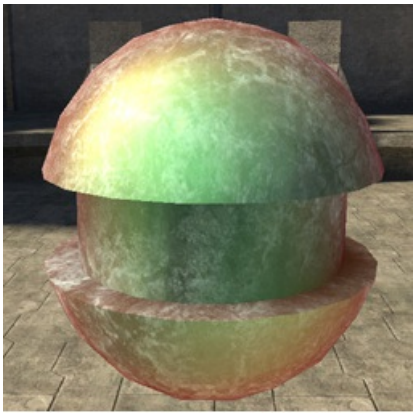


Figure 1 - normal_specular_power: 1

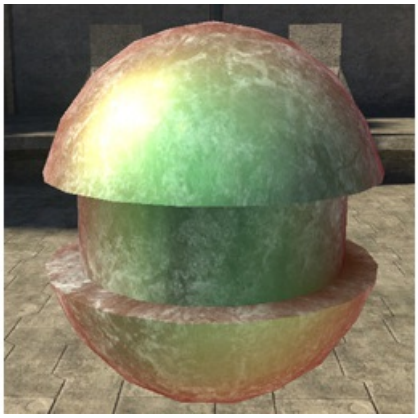


Figure 2 - normal_specular_power: 10 (default)

Normal Specular Tint

11/23/2022 • 2 minutes to read

The color of the specular lighting on mesh faces with normals pointing at the camera.



Figure 1 - normal_specular_tint: white (default)

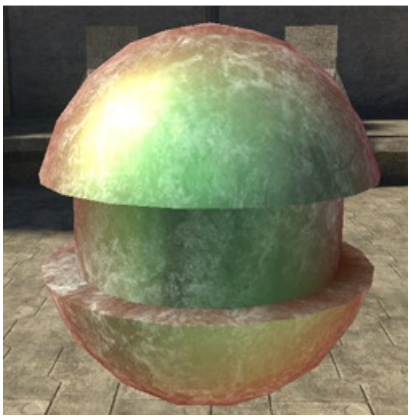


Figure 2 - normal_specular_tint: [0,255,0]

Occlusion Parameter Map

11/23/2022 • 2 minutes to read

Masks the ambient color AND the rim specular.



Figure 1 - occlusion_parameter_map with red ambient_tint

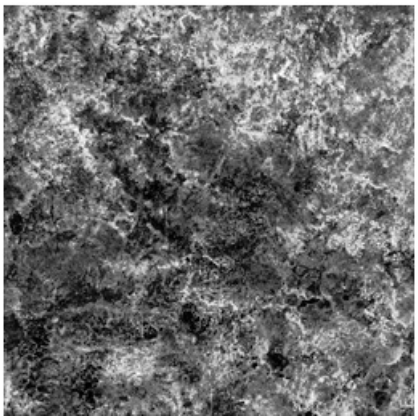


Figure 2 - Map used in Figure 1

Order 3 Area Specular

11/23/2022 • 2 minutes to read

Uses more of the lighting information in the scene to generate the specular component, so it's more accurate but also more expensive.



Figure 1 - order3_area_specular: off

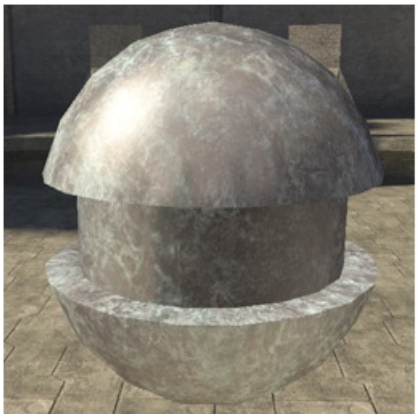


Figure 2 - order3_area_specular: on

Parallax Properties

11/23/2022 • 2 minutes to read

Parallax defines a special kind of bump mapping where pixels may occlude other pixels in the same map.

> **NOTE**
> This is an expensive option and should be used sparingly.

Parallax Mapping uses a specially formatted .bitmap tag called a **height map**. When you import a bitmap to use as a height map, either name it with a _height at the end, or make sure and select *Height Map* in the bitmap tag (and then re-import). The .bitmap parameter bump_height has nothing to do with the height of the parallax mapping. The height of the parallax is based solely on the green channel of the height_map.

> **IMPORTANT**
> Parallax mapping is a very subtle technique. If you set the scale too high, or have too much contrast in your bitmap when you import it, the parallax mapping will look ugly on your surface.

There are three options for Parallax

off

- **[no parameters]** — No parallax mapping. This is the default setting.

simple

- **height_map** [bitmap] — Differences in pixel values are used to determine placement and occlusion of pixels on the surface. Based on the differences in Value (lightness/brightness) between pixels in the green channel of the bitmap.
- **height_scale** [value]— The amount of displacement/occlusion applied to the affected pixels.

interpolated

The interpolated parallax map makes two corrections to the location of light and displacement of pixels across the surface and then averages between the two. This is more expensive than the standard parallax mapping option.

- **height_map** [bitmap] — Differences in pixel values are used to determine placement and occlusion of pixels on the surface. Based on the differences in value (lightness/brightness) between pixels in the green channel of the bitmap.
- **height_scale** [value] — The amount of displacement/occlusion applied to the affected pixels.

Rim Coefficient

11/23/2022 • 2 minutes to read

Strength of rim specular.



Figure 1 - rim_coefficient: 0.1



Figure 2 - rim_coefficient: 0.5



Figure 3 - rim_coefficient: 1 (default)



Figure 4 - rim_coefficient: 2

Rim Maps Transition Ratio

11/23/2022 • 2 minutes to read

Controls how much the rim specular is affected by the specular or diffuse maps.

- 1.0 = rim color from specular map
- -1.0 = rim color from diffuse
- 0 = rim color from rim parameters only.



Figure 1 - rim_maps_transition_ratio: 1



Figure 2 - rim_maps_transition_ratio: 0

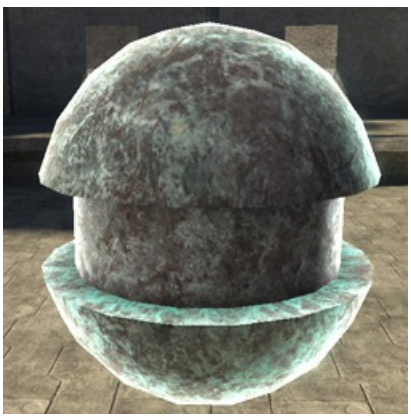


Figure 3 - rim_maps_transition_ratio: -1

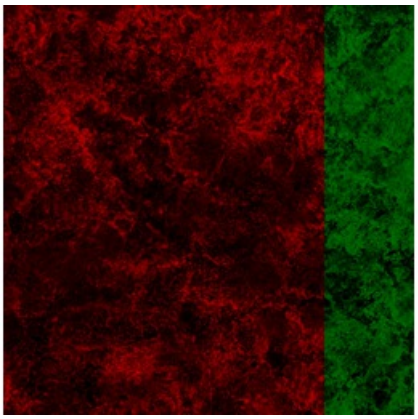


Figure 4 - Specular map used in figures 1 - 3

Rim Power

11/23/2022 • 2 minutes to read

Controls the overall balance of size and intensity in rim highlights. High numbers (5+) make small, hot highlights; lower numbers are broader and softer. Typical human values 1.5-3. Numbers lower than 1 are not advised.



Figure 1 - rim_power: 0.5



Figure 2 - rim_power: 1



Figure 3 - rim_power: 2 (default)



Figure 4 - rim_power: 4

Rim Start

11/23/2022 • 2 minutes to read

Controls how far the rim specular highlight wraps around the edge of the model. High values restrict highlight to very close to the edge, low values make broader highlights.



Figure 1 - rim_start: 0.1



Figure 2 - rim_start: 0.4



Figure 3 - rim_start: 0.7 (default)



Figure 4 - rim_start: 1

Rim Tint

11/23/2022 • 2 minutes to read

Tints the rim.

> **NOTE**
> The default of black needs to be changed to something else before you'll see any rim spec, regardless of what the other settings are



Figure 1 - rim_tint: white



Figure 2 - rim_tint: [0,255,0]

Roughness Properties

11/23/2022 • 2 minutes to read

Controls the size of the specular highlight, including the area and environment specular. A higher value results in light being spread out across the surface. A small value results in small, pinpoint light on the surface. For environment reflection, roughness controls the sharpness of the reflection map, so the higher the number, the blurrier the reflections.

Defined by a value from 0-1, but higher numbers can be entered for special effects.



Figure 1 - roughness: 0.1



Figure 2 - roughness: 0.5



Figure 3 - roughness: 1



Figure 4 - roughness: 2

Self-Illumination Properties

11/23/2022 • 4 minutes to read

Self-Illumination defines which pixels override object lighting.

There are six properties:

- off
- simple
- 3_channel_self_illum
- plasma
- from_diffuse
- illum_detail

off

- **[no parameters]**— No self-illumination. This is the default setting.

simple

- **self_illum_map** [bitmap] — Alpha channel of the selected bitmap controls which pixels are self-illuminated. The higher the color value in the alpha channel (of the self_illum_map), the more object lighting is overridden (255 = 100%, 0 = 0%).
- **self_illum_color** [color value] — Multiplied with the RGB channels of the self_illum_map to determine the color of light the object illuminates itself with.
- **self_illum_intensity** [numerical value] — the intensity of the light the object illuminates itself with.

3_channel_self_illum

- **sel_illum_map** [bitmap]— Alpha channel of the selected bitmap controls which pixels are self-illuminated. RGB channels of the selected bitmap are used to modulate between channels A, B, and C. The higher the color value in the alpha channel of the self_illum_map, the more object lighting is overridden (255 = 100%, 0 = 0%).
- **channel_a** [color] — The color of self-illuminated pixels from the Red channel of the self_illum_map.
- **channel_a_alpha** [value] — Governs the intensity of the self-illuminated pixels in the Red channel of the self_illum_map.
- **channel_b** [color] — The color of self-illuminated pixels from the Green channel of the self_illum_map.
- **channel_b_alpha** [value] — Governs the intensity of the self-illuminated pixels in the Green channel of the self_illum_map.
- **channel_c** [color] — The color of self-illuminated pixels from the Blue channel of the self_illum_map.
- **channel_c_alpha** [value] — Governs the intensity of the self-illuminated pixels in the Blue channel of the self_illum_map.
- **self_illum_intensity** [numerical value]— The intensity of the light the object illuminates itself with.

plasma

Plasma shaders take two bitmaps (noise maps) and compare them. Where values are the same (or close) between the two bitmaps, self-illumination is increased. Where values are different between the two bitmaps, self-illumination is decreased. For each area of similarity, three lines are created: sharp, medium, and wide—based on how similar the values are.

> **NOTE**
> Only the values placed in the Red channel of the bitmaps (noise_maps) are used for comparison.

- **noise_map_a** [bitmap]— Pixels which are of similar value to those of noise_map_b will be self-illuminated.
- **noise_map_b** [bitmap]— Pixels which are of similar value to those of noise_map_a will be self-illuminated.
- **color_medium** [color]— The color of the medium sized line. Color_medium is a blend between the sharp and wide lines.
- **color_medium_alpha** [value]— Governs the intensity of the self-illumination for the medium sized line.
- **color_wide** [color]— The color of the wide line. Used where the difference in value between the pixels in the noise maps is greatest.
- **color_wide_alpha** [value]— Governs the intensity of the self-illumination for the wide (or large) line.
- **color_sharp** [color]— The color of the sharp line. Drawn where the difference in pixel value between the noise maps is the smallest.
- **color_sharp_alpha** [value]— Governs the intensity of the self-illumination for the sharp (or small) line.
- **self_illum_intensity** [numerical value]— The overall intensity of the light the object illuminates itself with.
- **alpha_mask_map** [bitmap]— Uses the alpha channel of the specified bitmap for per-pixel control of the plasma illumination. 255 (white) does nothing, any value less than 255 will decrease the plasma illumination.
- **thinness_medium** [value] — Governs the thinness of the color_medium area. A higher number will cause the color_medium line/area to become smaller (thinner). A low number will spread out the color_medium area of the shader.
- **thinness_wide** [value] — Governs the thinness of the color_wide area. A higher number will cause the color_wide line/area to become smaller (thinner). A low number will spread out the color_wide area of the shader.
- **thinness_sharp** [value] — Governs the thinness of the color_sharp area. A higher number will cause the color_sharp line/area to become smaller (thinner). A low number will spread out the color_sharp area of the shader.

from_diffuse

Self-illumination of pixels is governed by the values of the RGB channels of the base_map from the albedo properties of the shader. 0 = no self-illumination. 255 = full self-illumination.

- **self_illum_color** [color]— The color of self-illuminated pixels.

- **self_illum_intensity** [numerical value]— The intensity of the light the object illuminates itself with.

illum_detail

- **self_illum_map** [bitmap]— The alpha channel of the selected bitmap is multiplied with the alpha channel of the self_illum_detail_map to control which pixels are self-illuminated. The higher the color value of each pixel in the alpha channel (of the self_illum_map), the more object lighting is overridden (255 = 100%, 0 = 0%).
- **self_illum_detail_map** [bitmap]— The RGB channel of the selected bitmap is double-multiplied with the self_illum_intensity setting. 50% gray does nothing, darker than 50% gray decreases the intensity. The alpha channel of the self_illum_detail_map is multiplied with the alpha channel of the the self_illum_map to control which pixels are self-illuminated.
- **self_illum_color** [color]— Multiplied with the RGB channels of the self_illum_map to determine the color of the light the object illuminates itself with.
- **self_illum_intensity** [value]— The intensity of the light the object illuminates itself with.

Specular Coefficient Properties

11/23/2022 • 2 minutes to read

Makes the shader more or less shiny.

Defined by a value from 0-1, but higher numbers can be entered for special effects. A value of zero will result in no specular at all.

The important thing to keep in mind is that this value contols overall specularity.



Figure 1 - specular_coefficient: 0.1



Figure 2 - specular_coefficient: 0.5



Figure 3 - specular_coefficient: 1



Figure 4 - specular_coefficient: 2



Figure 5 - specular_coefficient: 5

Specular Map

11/23/2022 • 2 minutes to read

Gives you per-pixel control over both specular strength and color. If the map is greyscale, it only affects strength, not color. Use and RGBA map to control color (and in this case, the alpha channel would control strength).



Figure 1 - with specular_map

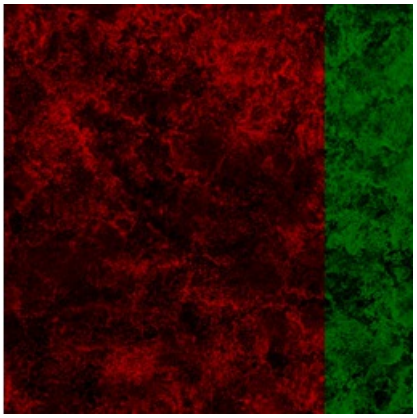


Figure 2 - specular_map

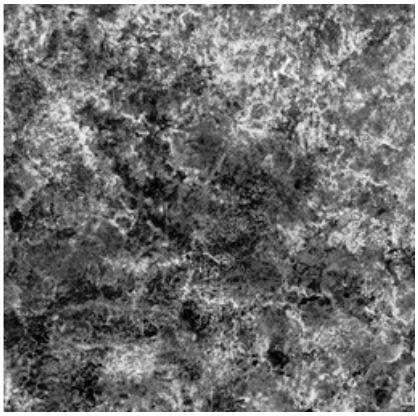


Figure 3 - specular_map alpha



Figure 4 - without specular_map

Specular Mask

11/23/2022 • 2 minutes to read

Specular Mask defines how specular properties are controlled on a per-pixel basis.

> **NOTE**
> The specular properties (if any) are defined in the Material Model, this option simply defines where to find the mask. Regarding the grayscale values in the alpha channel (if used), white is maximum specularity as defined in the material_model properties, black is zero specularity.

There are three options for specular mask parameters:

no_specular_mask

- **[no parameters]** — No per-pixel control of specular properties. This is the default setting. This will give you a constant specular highlight across the mesh, resulting in a very plastic look.

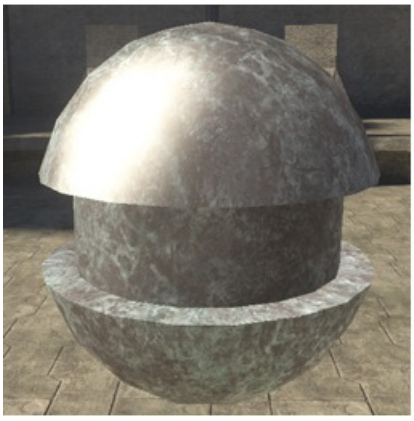


Figure 1 - no_specular_mask

specular_mask_from_diffuse

- **[no parameters]** — Uses the alpha channel of the base_map in the albedo properties for per-pixel control of specular properties. This option (and the one below) let you break up the specular highlight for a more realistic look.



Figure 2 - specular_mask_from_diffuse

specular_mask_from_texture

- **specular_mask_texture** [bitmap] — Uses the red channel of the material_texture as defined in the cook_torrance material_model.

Specular Power

11/23/2022 • 2 minutes to read

Similar to roughness, except in reverse.

Controls the overall balance of size and intensity in specular highlights. High numbers (15+) make small, hot highlights— lower numbers are broader and softer. Typical human values are 4-9.



Figure 1 - specular_power: 1



Figure 2 - specular_power: 5

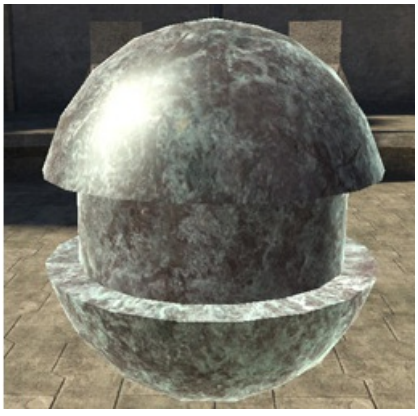


Figure 3 - specular_power: 10 (default)



Figure 4 - specular_power: 20

Specular Tint

11/23/2022 • 2 minutes to read

Tints the specular highlight with a color.



Figure 1 - specular_tint: white



Figure 2 - specular_tint: red

Standard Propterties

11/23/2022 • 2 minutes to read

Shaders are added on a per-material basis to 3D objects.

Sections of the shader are revealed or hidden based on the category options you choose.

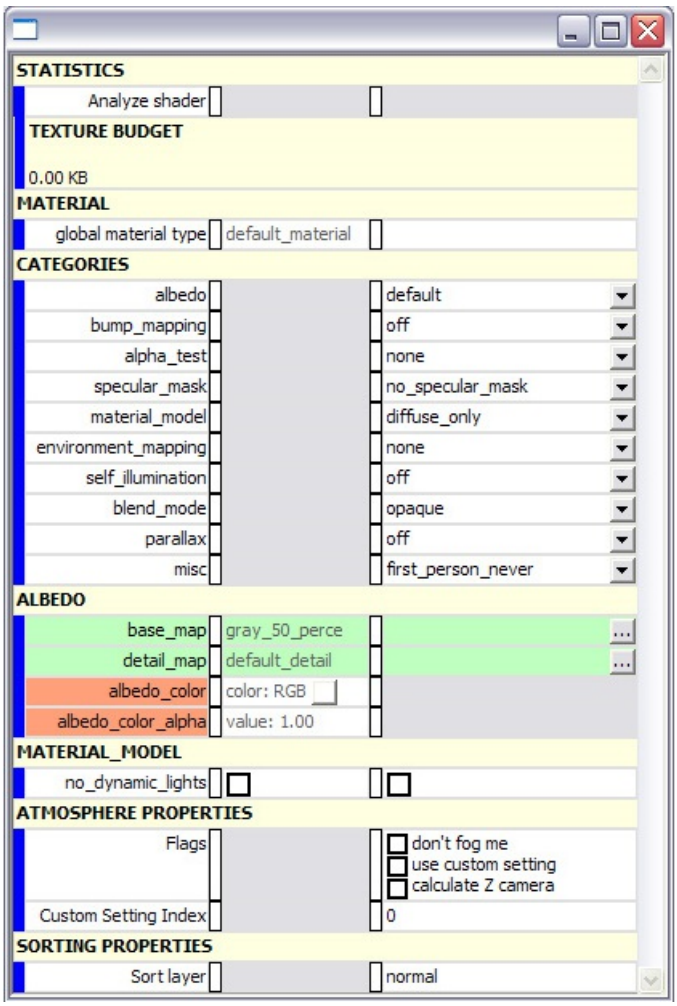


Figure 1 - Shader Shader

Statistics

- Analyze shader— Currently not used. Meant to give you a total cost for the shader so that you can see the effect of changes without having to load it in the game.
- Texture Budget (not used)

Material

- global material type [string, enter manually] — default: default_material

Defines which sounds and effects to use with this shader in the game.

Categories

- albedo [drop down list] — default: default (pronounced Al-Bee-Doh) Defines the surface color. Also

known as diffuse.

- bump_mapping [drop down list] — default: default Defines variations in how lighting affects the surface.
- alpha_test [drop down list] — default: none Defines whether a pixel should be rendered or ignored.
- specular_mask [drop down list] — default: no_specular_mask Defines how specular properties are controlled on a per-pixel basis.
- material_model [drop down list] — default: diffuse_only Specify the type of shader.
- environment_mapping [drop down list] — default: none Define the source used for the reflective environment mapping.
- self_illumination [drop down list] — default: off Defines which pixels over-ride object lighting, also known as glow.
- blend_mode [drop down list] — default: opaque Defines how the shader blends with the objects behind it.
- parallax [drop down list] default: off Defines a special kind of bump mapping where pixels may occlude other pixels in the same map. Simulates the effect of displacement in a 2D shader.

Albedo

- See the albedo page.

Material_Model

- See the material_model page.

Atmosphere Properties

- Flags [on/off checkboxes]— default: all off
- calculate Z camera:
- Custom Setting Inde [number]— default: 0

Sorting Properties

- Sort layer [drop down list]— default: normal

Subsurface Coefficient

11/23/2022 • 2 minutes to read

Turns the subsurface lighting on and off.

- 0 = No subsurface lighting.
- .75 = Typical values.
- 1.0 = Subsurface lighting as strong as normal diffuse lighting.



Figure 1 - subsurface_coefficient: 0.1



Figure 2 - subsurface_coefficient: 0.5

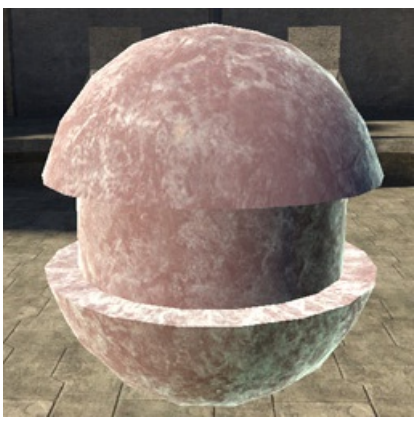


Figure 3 - subsurface_coefficient: 1



Figure 4 - subsurface_coefficient: 2

Subsurface Map

11/23/2022 • 2 minutes to read

Masks subsurface lighting.



Figure 1 - With subsurface_map

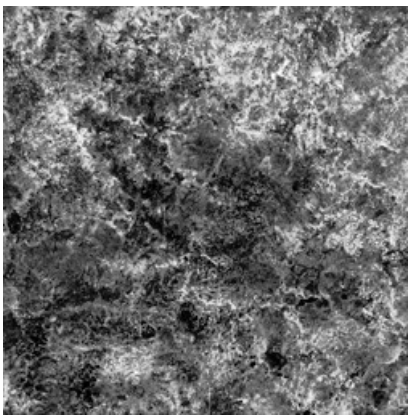


Figure 2 - Map



Figure 3 - Without subsurface_map

Subsurface Propagation Bias

11/23/2022 • 2 minutes to read

Controls size of the subsurface effect.

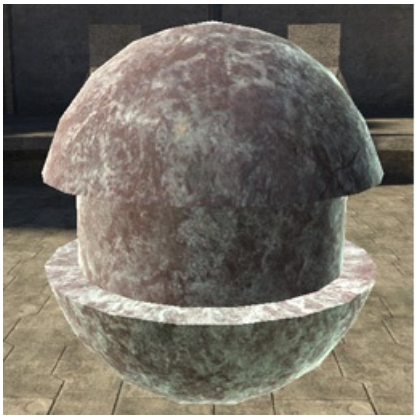


Figure 1 - subsurface_propagation_bias: 0



Figure 2 - subsurface_propagation_bias: 0.5



Figure 3 - subsurface_propagation_bias: 1



Figure 4 - subsurface_propagation_bias: 2

Subsurface Tint

11/23/2022 • 2 minutes to read

Tints the subsurface lighting.



Figure 1 - subsurface_tint: white (default)



Figure 2 - subsurface_tint: [192,108,108]

Templates

11/23/2022 • 2 minutes to read

Create Shader Template

A shader tag can reference another shader tag and inherit values from it. Part of this process includes letting artists freely decide which category_options and parameter values of a shader will be inherited by other shaders.

Here is the process:

Check the template me checkbox (see Figure 1).

A column of checkboxes will show up in the Categories block. Check the category_options which you want to lock in the template.

If one category is locked, a column of checkboxes will show up in corresponding parameter lists (like categories of albedo and material_mode shown in Figure 1).

Check the parameters you wish to lock.

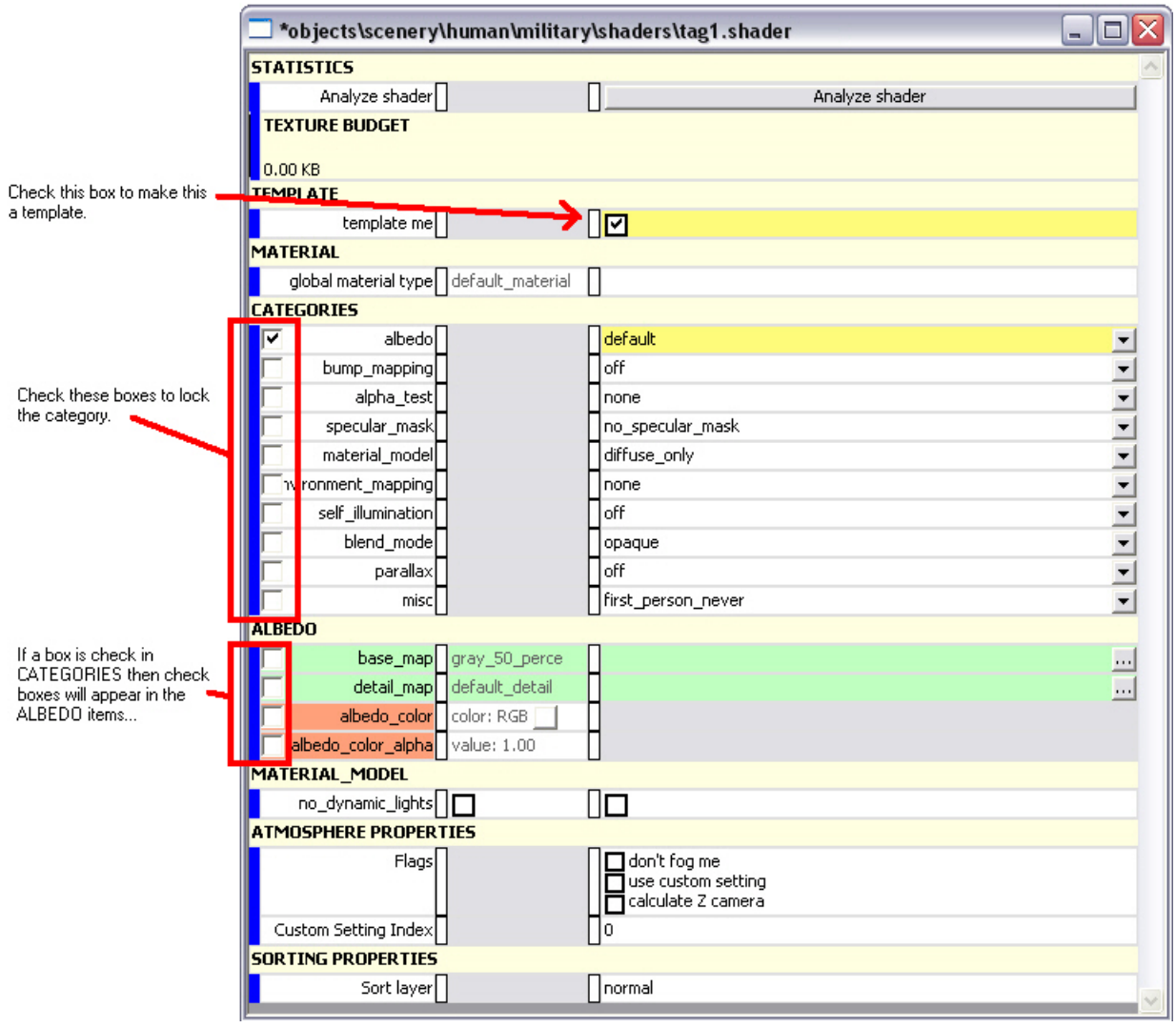


Figure 1 - Creating a shader template

Refer to Shader Template

The shader refers to a template by the template_reference line in the Material block.

The following are inherited from the template (see Figure 2):

material_name

global_material_type

All default values of catergory_options (including both locked and unlocked fields)

Shader Views

There are two different views of shaders: Full and compact (see Figures 2 and 3). Compact view is the default view of a derived shader. In compact view, all locked parameters are hidden, except those which have been modified. In full view, all fields are displayed but the locked fields are indicated by yellow background color.

> **NOTE**
> The locked fields are still editable if the shader tag is checked out.

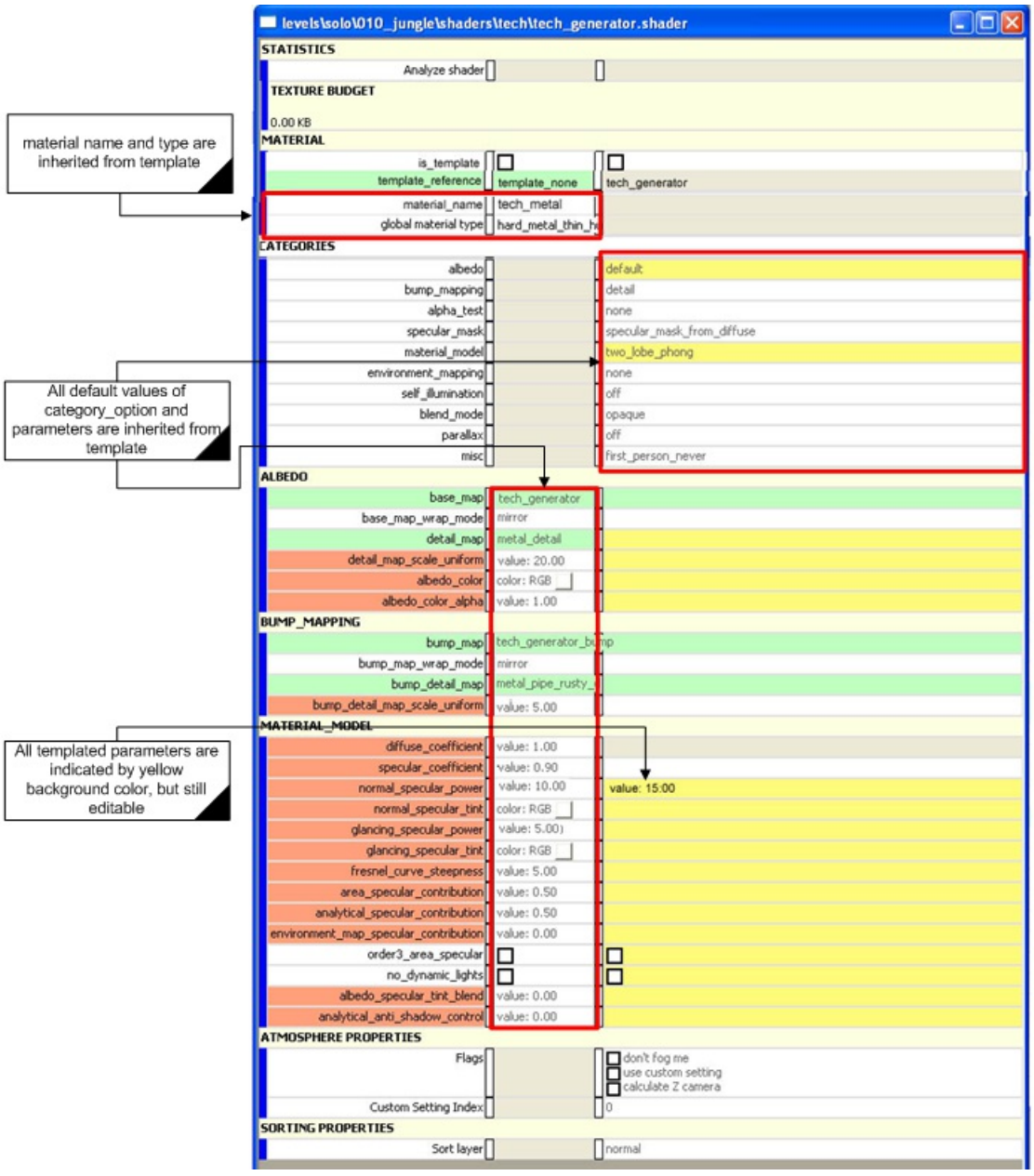


Figure 2 - Refer to a shader template

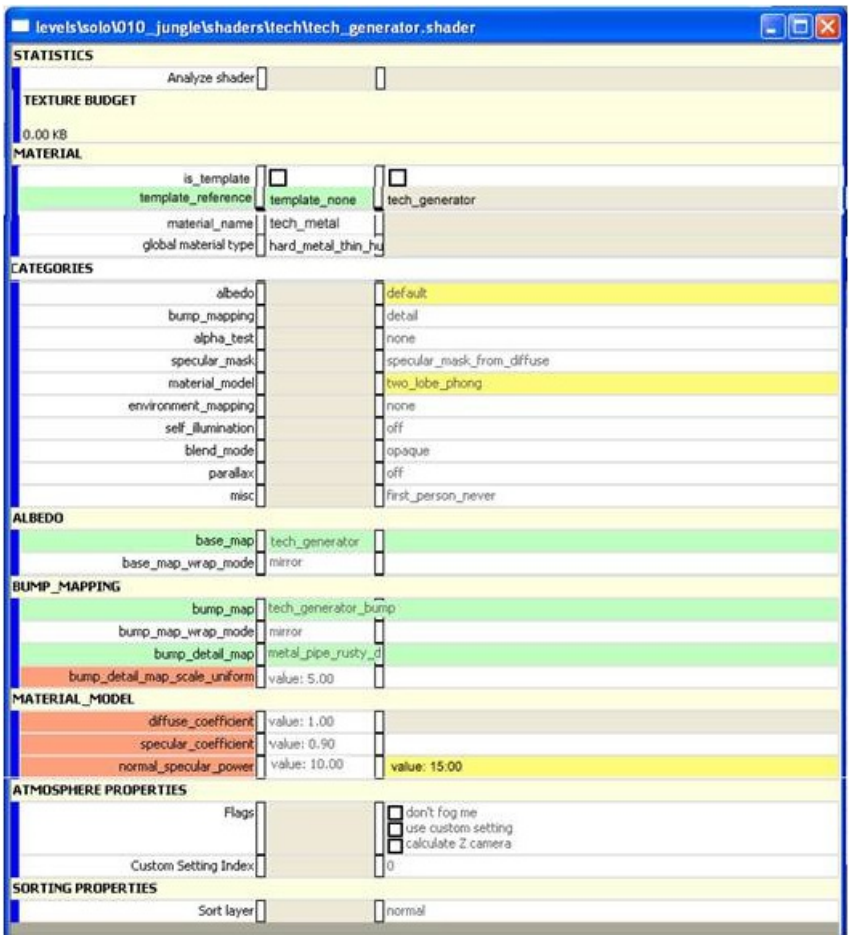


Figure 3 - Compact view of Shader

Notes About Templates

- We only support one layer of inheritance of shader templates to simplify the system, which means one template can not refer to another - template
- The shader template can also act as a normal shader which can be directly applied to objects for rendering
- Template data becomes the default values in the shader tag, so to revert back to the template value delete your shader override value

Terrain

11/23/2022 • 2 minutes to read

Categories

- albedo
- bump_mapping
- alpha_test
- specular_mask
- material_model
- environment_mapping
- self_illumination
- blend_mode
- parallax

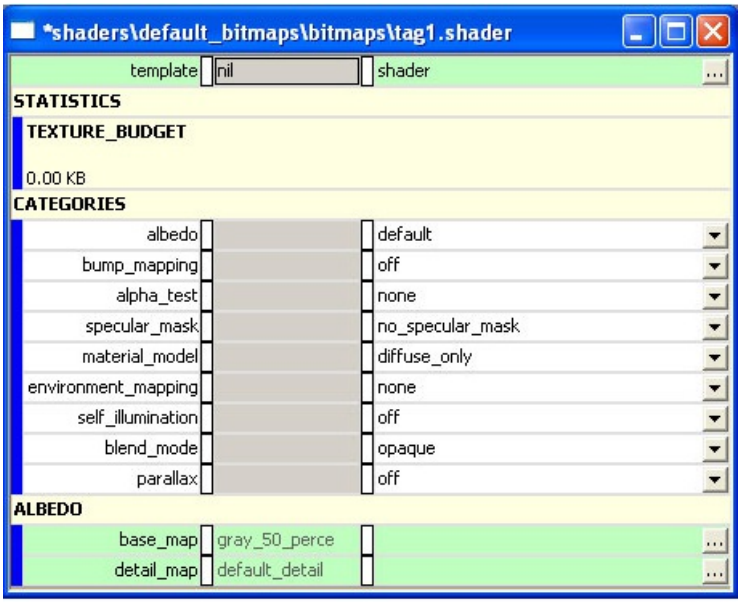


Figure 1 - A sample .shader tag with default settings

Template

It's not necessary to select a specific template for shaders in Halo 3— there is a single template (format for a shader tag) that covers all types of shaders.

Statistics

- **Texture_budget** — Lists how many kilobytes of memory the current configuration of the shader would use up in-game.

Transparence Coefficient

11/23/2022 • 2 minutes to read

Turns "transparence" lighting on and off. Transparence lighting uses reversed surface normals, so it lights through the model.

- 0.0 = no effect
- 1.0 = transparence lighting as strong as the diffuse lighting.



Figure 1 - transparence_coefficient: 0.1



Figure 2 - transparence_coefficient: 0.5



Figure 3 - transparence_coefficient: 1



Figure 4 - transparence_coefficient: 2

Transparence Map Properties

11/23/2022 • 2 minutes to read

Gives you per-pixel control over both transparence strength.



Figure 1 - With transparence map

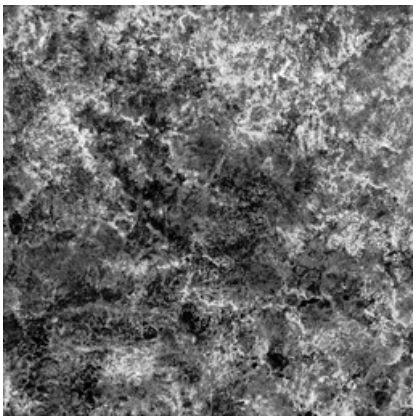


Figure 2 - Transparence map used



Figure 3 - Without transparence map

Transparence Normal Bias

11/23/2022 • 2 minutes to read

Forces transparence lighting to the edges or center of the model.

- -1.0 adds transparence lighting to the inside of the model
- 1.0 adds to the rims only
- 0 does the whole model.



Figure 1 - transparence_normal_bias: -1



Figure 2 - transparence_normal_bias: 0



Figure 3 - transparence_normal_bias: 1

Transparence Tint Properties

11/23/2022 • 2 minutes to read

Tints the transparence effect.



Figure 1 - transparence_coefficient: white (default)



Figure 2 - transparence_coefficient: red

Dynamic Lights

11/23/2022 • 8 minutes to read

Dynamic lights use a .light tag and can be set up for use in the game by placing them in 3ds Max (with a marker), placing them in Sapien, or by attaching them to an object via a marker (in 3ds Max). They work independently of the lightmap (so lightmaps are not necessary for dynamic lights to function correctly).

Placing Dynamic Lights in 3ds Max

Dynamic lights can be placed in a scenario using a Marker in 3ds Max which then gets picked up by the importer and hooked up to a .light tag within a subdirectory of your scenario. The marker names used in Max need to use one of the following formats:

NAME FORMAT	EXAMPLE
#<name>_light<numbers>	#green_light27
#<name>_light<numbers>	#green_light27
#light_<name><numbers>	light_red42
#light_<name><numbers>	light_red42

The importer searches through all subdirectories within your level directory, so you can place your light tag anywhere within that folder (not just in a "lights" directory). If the importer fails to find the .light tag you named in Max, it looks in levels\shared\lights and if it doesn't find it there, it assigns it a default name and location (even though it doesn't exist) of levels\<levelname>\lights\<name>.light.

Step-by-step instructions:

1. Open your level in 3ds Max.
2. Place a marker at the location you'd like the dynamic light to show up in game.

> **NOTE**
> The dynamic light will point in the direction of the x-axis (local axis) on the marker.

3. Name the marker geometry using the format listed above.
4. Save and Export.
5. If it doesn't already exist, create the .light tag for the dynamic light you want to use (see below for more information on creating .light tags).
6. Import your Level using Tool (from within Guerilla).
7. Xsync and Launch on the 360.

Placing Dynamic Lights in Sapien

There is currently a bug which doesn't allow Sapien to edit the properties of the .light tag. Until it is fixed, you'll have to use Guerilla to edit the properties of your light and use Sapien only for positioning and placement.

Here's how to place dynamic lights in your scenario using Sapien:

1. Open your scenario in Sapien (make sure it's checked out or writable)
2. In the *Hierarchy View*, click the **Edit Types** button.
3. From the *Object Class* drop down menu, select **Light** and click the **Add** button.
4. In the *Open* dialog that appears, navigate to the .light tag you want to place in the scenario. Once you've found it, click it to select it and then click the **Add Tags** button.
5. Click **Done** to close the dialog, then click **OK** to close the *Edit Types* dialog.
6. Go back to the *Hierarchy View*. Expand the *Scenario* folder and then expand the *Lighting Data* folder.
7. Click the **Lights** folder.
8. Right click in your scenario to place an instance of a light. You can move or rotate it to the exact place you want it using the move gizmo.
9. In the *Properties Palette*, find the Type drop down list and select the name of the light tag you added to the scenario. Your light should appear in the game window.
10. You can also name your light, if you want, but that is all you need to do to get one set up (see Figure 1)

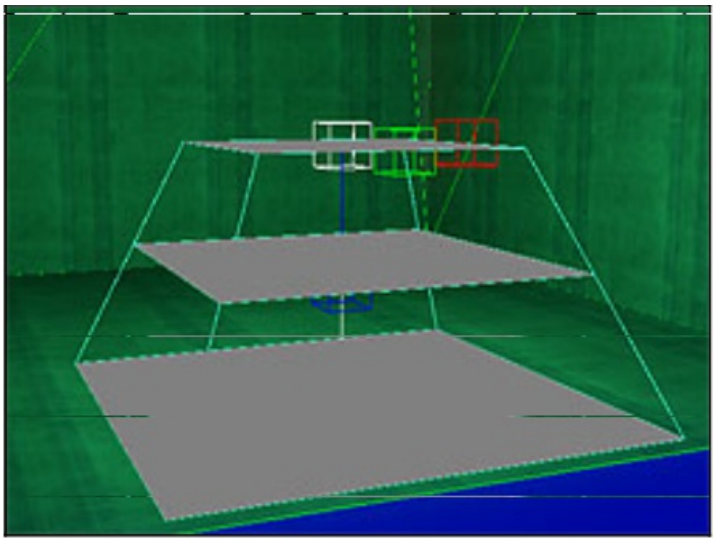


Figure 1 - A Dynamic Light placed in Sapien

Attach a Dynamic Light to an Object

Here are the steps for attaching a dynamic light to an object:

1. In 3ds Max, open the object you want to attach your light to (make sure it is checked out or writable).
2. Create a marker which is in the location and points to the direction you want your light to appear. For example, if you're trying to create muzzle flash, you would place a marker at the barrel of the gun pointing outwards. To create a marker, do the following:
 - Create some simple geometry to act as your marker (a nice little arrow pointing in the direction you want the light to display would be good).
 - Place a # at the start of the name of the marker geometry/node. So it would be called #marker. See Figure 2 for an example.



Figure 2 - The battle rifle model with markers (small red arrows) for locations where lights or effects will play.

3. Export and import your object.
4. Create a .light tag that you want to use for your dynamic light (see the .light tag reference below for more information on creating a .light tag).
5. Check out and open the object type tag for your object (either .scenery, .crate, .vehicle, .weapon, etc.).
6. Find the tag block labeled Attachments. Click the Add button to add a new attachment (your .light tag).
7. Click the Browse Button next to the Type box. Find your .light tag and click OK to link it to the object tag.
8. In the Marker text box, type the name of the marker you set up in 3ds Max (without the # sign).
9. Save your object tag and Xsync. Your light should shine from the marker you specified.

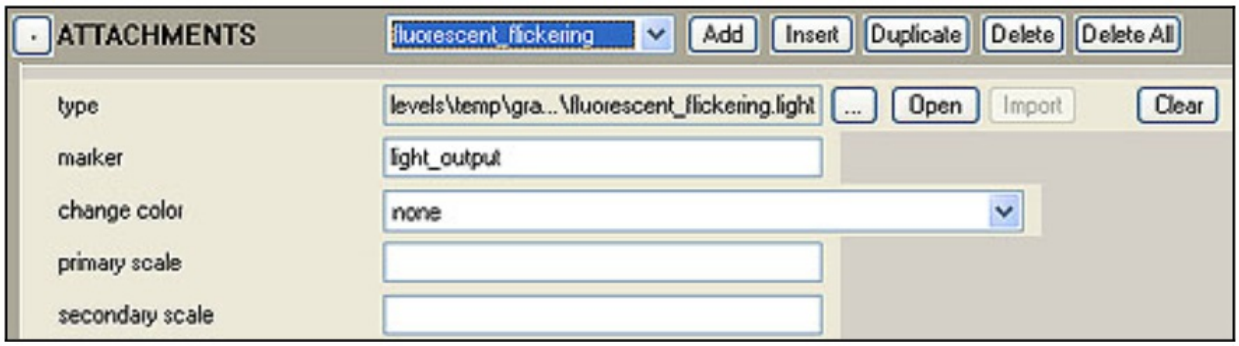


Figure 3 - The Attachments tag block of an object tag correctly set up for a .light attachment.

.light Tag Reference

Cheap Dynamic Lights vs. Expensive Dynamic Lights

	SHADOWS?	FRUSTRUM TYPE	CONFIGURABLE FRUSTRUM WIDGE AND HEIGHT?	LIGHT GELS?	PASSES TO RENDER
Cheap	No	Cone	No	No	1
Expensive	Yes	Square	Yes	Yes	3

- Flags
 - **Expensive Light**— Expensive dynamic lights can cast shadows, have square frustums (which can be re-sized), and can have gels applied to them. They also take 3 passes to render so they're very expensive compared to the cheap lights (which render on the same pass as the lightmap). See the table above which compares cheap and expensive lights.
 - **Shadow Casting**— Light will cast shadows on objects that come within range and will be blocked by environment geometry. This flag can only be enabled when the Expensive Light flag is checked.

Otherwise, it will not be available.

- **Render First Person Only**— Only renders the light in first-person view. Hasn't been implemented yet for Halo 3. Article will be updated when it is working.
- **Render Third Person Only**— Only renders the light in third-person view. Hasn't been implemented yet for Halo 3. Article will be updated when it is working.
- **Render Splitscreen**— Will render the dynamic light while splitscreen is on. Hasn't been implemented yet for Halo 3. Article will be updated when it is working.
- **Render While Active Camo**— Will render the light while the player is wearing active camo. Hasn't been implemented yet for Halo 3. Article will be updated when it is working.
- **Render in Multiplayer Override**— Will render the light while in multiplayer mode (dynamic lights default to "Off" otherwise). Hasn't been implemented yet for Halo 3. Article will be updated when it is working.
- **Move to Camera in First Person**— Places the light as if it were coming from the camera when in first person view— no matter where the light is actually placed in third person. A good example of how this is used is the flashlight. It has a different location in third person than what the player sees in first person.

- **Shape**
 - **Type**— Select either Frustum or Sphere. A sphere casts light (up to the maximum distance set) outward in all directions (like a standard incandescent lightbulb). A frustum light can be constrained to a particular direction and field of view.
 - **Maximum Distance**— The farthest distance that the light will travel.
 - **Frustum Near Width** — The Width of the beam for a frustum light (in world units). A large number creates a wider beam while a small number creates a narrow beam. This flag is only enabled when the Expensive flag is checked above.
 - **Frustum Height Scale** — Controls the aspect ratio (or shape) of a frustum light. Only available for editing with expensive lights (Expensive flag must be checked (see above)).
 - **Frustum Field of View**— The angle (in degrees) which light will be allowed to be cast from the source of a frustum light. Light is cast along the X-axis of the marker by which the light was placed.
- **Color** — Set the color to a basic single color, or you can set up a function which changes between up to four different colors.
- **Intensity** — Set the intensity or strength of the light. It can have a constant intensity, or can use functions, or be connected to an object via a function.
- **Falloff**
 - **Distance Diffusion** — The "size" of the light source. Pick a large number for a pinpoint source of light, or a small number for a wider light source.

> **NOTE**
>
> A larger light source will also increase the power of the light, so you may have to adjust intensity as well. See Figure 4 for examples.

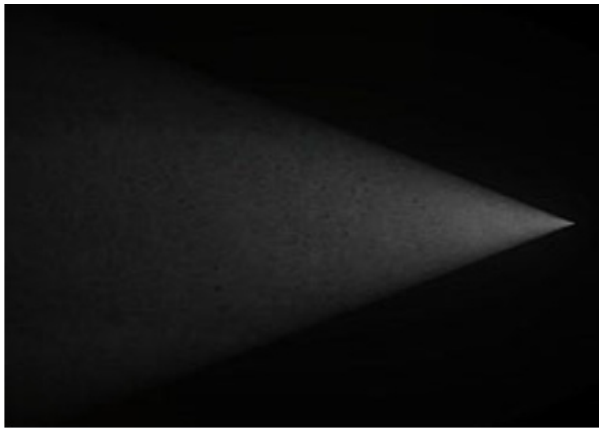
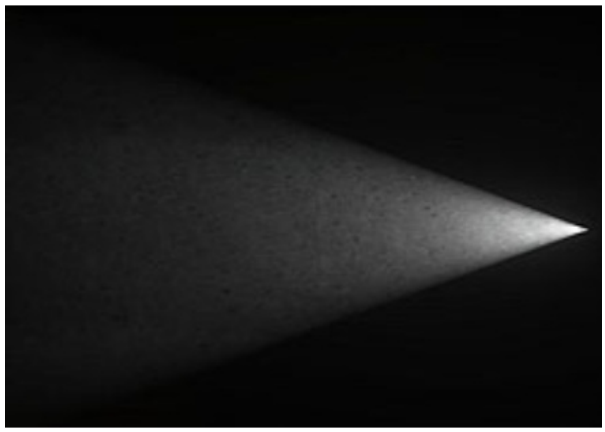
DISTANCE DIFFUSION = 1	DISTANCE DIFFUSION = 0.1
	

Figure 4 - The figure on the left has distance diffusion set to 1, the one on the right has distance diffusion set to 0.1. All other properties are the same.

- **Angular Smoothness**— The amount of smoothing done to the edge of the light beam to fade it in or out. A small number will give the light a harder edge, while a large number will give the light a soft edge. See Figure 5 below.

> **NOTE**
> When the light type is set to sphere, this setting is ignored.

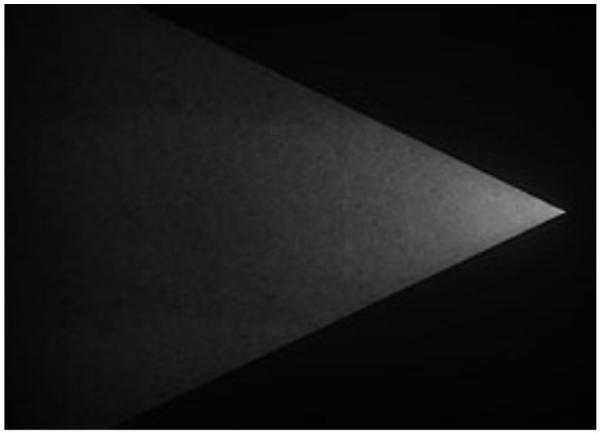
ANGULAR SMOOTHNESS = 0.2	ANGULAR SMOOTHNESS = 8
	

Figure 5 - The figure on the left has angular smoothness set to 0.2, the one on the right has angular smoothness set to 8. All other properties are the same.

- **Percent Spherical** — This number controls the amount (and thereby intensity) of light that will be emitted in all directions — not just the direction you have your frustum light aimed. Setting your light type to Sphere will negate any settings you place here. See Figure 6 below.

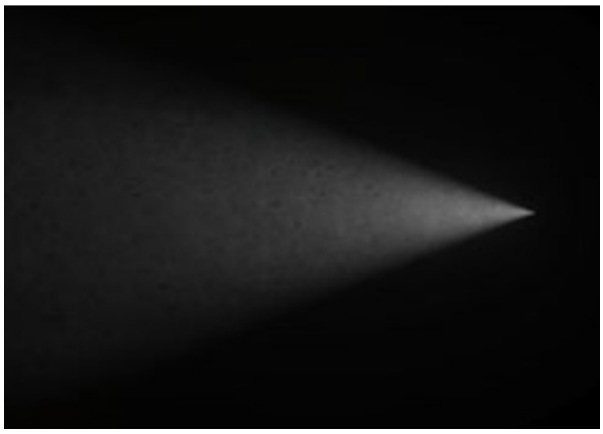
DISTANCE DIFFUSION = 1	DISTANCE DIFFUSION = 0.1
	

Figure 6 - The figure on the left has percent spherical set to 0, the one on the right has percent spherical set to 0.2. All other properties are the same.

- **Lifetime**
 - **Destroy Light After**— The time a light will be rendered (in seconds) before being destroyed, removed, and no longer rendered.
- **Distance to Fading Parameters**
 - **Illumination Fade Distance**— The general distance from the light at which the player needs to be before it fades out. Settings are: Fade very far, fade far, fade medium, fade close, fade very close.
 - **Shadow Fade Distance**— The general distance from the light at which the player needs to be before the light stops rendering shadows. Settings are: Fade very far, fade far, fade medium, fade close, fade very close. Shadows aren't currently functioning for dynamic lights, so this doesn't have any effect.

BSP

11/23/2022 • 3 minutes to read

Multiple BSP Environments

It's important to set up a large environment with multiple BSPs to conserve resources when the map's in play. For example, if a player goes from area A to B then to C, area A can drop out saving resources.

Here are some important notes to consider when setting up an environment with multiple BSPs.

Create a Selection Set

A best practice is to create a selection set that includes geometry, seams, and the .ASS files.

Facing and Naming

BSPs inside another face in two directions, both out and in. Those that are facing out need to be named +seam:a, and those that face in +seam:b. See Figure 1 which shows a bsp (blue) inside an outer bsp (black).

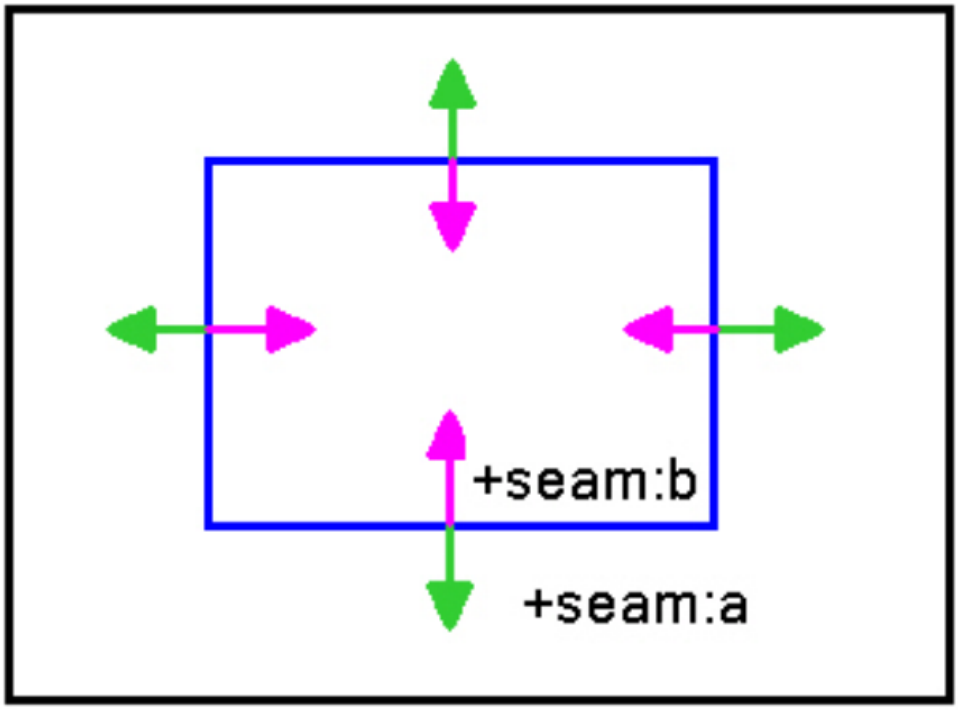


Figure 1 - Naming and Facing

Seams

Workflow Pattern

- Make seam changes and ensure:
 - The seam is in the relevant BSP selection set
 - The normals point into the BSP
 - Edge verts line up with edge BSP geometry
 - The tessellation is the same as its immediate seam neighbor
 - The seam geometry has the name of the BSP relevant to it in the

 - Object name
 - Material
- Delete extraneous .ass files from the structure directory
- Export changed BSPs
- Check out structure_seams and scenario tags
- Run the tool command structure_seams
 - The output log should tell you that you have some number of matching seams.
- Re-import changed BSPs

BSP creation glossary

Matching Seams

- Every seam has a matching seam — the two matching seams need to have the same triangulation
- The seams need to be coincident with BSP vertices with opposite facing normals

Sub Material Naming

- Each seam that defines a BSP will have its own submaterial
- The seam material name must match the eventual BSP name

 Example:

 - +seam:030_BSP_000

 Apply the above material name to the seams that define BSP_000 and when you export the ASS file make sure to name it 030_BSP_000

Poop Exporting

- Each poop should be exported only once
- Choose a BSP to associate the poop with and export it with that BSP only

> **NOTE**
> A poop that straddles a BSP seam cannot penetrate into the second cluster of a BSP from that seam as defined by your portals

Structure_Seam Tags

- *.structure_seams files are a compiled version of the structure seams for a level. When a structure imports it uses this tag for mapping the seams of structures to seam in the *.structure_seams file. Then when two structures match the same seam in the *.structure_seams file we know that they are connected by that seam." (it should be noted that the tag, structure_seams, for the relevant scenario, needs to be checked out).

Debugging

If your debug output reports seam leakage you will need to fix that before debugging.

- Any BSP edges outside of your exported BSP (remember you can have an organic terrain that stretches outside of your defined BSP) that are not directly connected to the BSP will generate open edges
- Seams do not make a manifold BSP and therefore the importer does not throw out triangles outside of

the BSP— it's a surface walking process instead

Commands

- debug_structure_seams is on by default. This will show you mismatched clusters across seams and error geometry from the import of the structure_seams tag
- debug_structure_seam_edges is useful for rendering the recognized edge border between BSPs
- structures_from_scenario imports all relevant/changed BSPs for that scenario for you – getting around the incredibly manual nature of this process. Obviously, you need the relevant BSP files (ass, structure_BSP, structure_lighting_info) files checked out for this to work.

> **NOTE**
>
> Reimporting structure_seams invalidates the seams in your current structure_BSP tags. Therefore it should be done before importing BSPs. In fact you only need to run this if the seam geometry between BSPs changes. If seams are mismatched you will see error geometry by setting the debug_structure_seams hs variable, which is now on by default.

Debug

11/23/2022 • 2 minutes to read

Debug Menu

Information on the debug menu and how you can create your own.

Profile Settings Commands

Debug commands that can be used to modify player profile settings in all but a shipping build of the game.

Debug Menu

11/23/2022 • 2 minutes to read

The Basics

- The debug_menu_init.txt file is located in your \main\bin directory.
- You can create a custom debug menu, called debug_menu_user_init.txt which will append to the default debug menu (but which will still allow you to get the latest versions of the default menu without erasing your custom items). See below for instructions.

Create a custom debug menu

- If the standard debug menu doesn't suit your needs, you can create a custom debug menu just for yourself, but which will still allow you to get the latest versions of the default menu without erasing your custom items. The custom menu simply appends to the bottom of the default one.

Here's how to set it up:

1. Create a text file named **debug_menu_user_init.txt**.
2. Using the same format as that found in the debug_menu_init file, place the commands you want in your menu in the file you just created.
3. Save the file and copy it to the bin folder. Ex: H3EK\bin
4. If you're running the game when you copy the file over, you'll have to restart the game to see your changes to the menu.

Profile Settings Commands

11/23/2022 • 2 minutes to read

The following debug commands can be used to modify player profile settings in all but the shipping build of the game and they will also save profile as they are made. You can use these commands to change your settings but they won't be saved to your profile (and you'll then have to enter these commands every time you launch the game).

These commands will no longer work from your init.txt file because there is a delay between the time init.txt is read and when the signed-in users are detected by the game.

- **controller_set_look_inverted** [controller] [true|false]— Sets look inversion for specified controller, where controller is one of [controller1, controller2, controller3, controller4]
- **controller_set_vibration_enabled** [controller] [true|false]— Sets vibration for specified controller, where controller is one of [controller1, controller2, controller3, controller4]
- **controller_set_flight_stick_aircraft_controls** [controller] [true|false]— Sets airrcraft flight stick controls for specified controller, where controller is one of [controller1, controller2, controller3, controller4]
- **controller_set_auto_center_look** [controller] [true|false]— Sets auto center look for specified controller, where controller is one of [controller1, controller2, controller3, controller4]
- **controller_set_button_preset** [controller] [preset]— Sets button preset for specified controller, where controller is one of [controller1, controller2, controller3, controller4] and preset is one of [standard south_paw boxer green_thumb]
- **controller_set_joystick_preset** [controller] [preset]— Sets joystick preset for specified controller, where controller is one of [controller1, controller2, controller3, controller4] and preset is one of [standard south_paw legacy legacy_south_paw]
- **controller_set_look_sensitivity** [controller] [value]— Sets look sensitivity for specified controller, where controller is one of [controller1, controller2, controller3, controller4] and value is some number in the range of [0,9] inclusive
- **controller_unlock_single_player_levels** [controller]— Unlocks all single player levels for specified controller, where controller is one of [controller1, controller2, controller3, controller4]
- **controller_set_primary_change_color** [controller] [color]— Sets primary change color for specified controller, where controller is one of [controller1, controller2, controller3, controller4] and color is one of [white steel red orange gold olive green sage cyan teal cobalt blue violet purple pink crimson brown tan]
- **controller_set_secondary_change_color** [controller] [color]— Sets secondary change color for specified controller, where controller is one of [controller1, controller2, controller3, controller4] and color is one of [white steel red orange gold olive green sage cyan teal cobalt blue violet purple pink crimson brown tan]
- **controller_set_tertiary_change_color** [controller] [color]— Sets tertiary color for specified controller, where controller is one of [controller1, controller2, controller3, controller4] and color is one of [white steel red orange gold olive green sage cyan teal cobalt blue violet purple pink crimson brown tan]
- **controller_set_quaternary_change_color** [controller] [color]— Sets quaternary change color for specified controller, where controller is one of [controller1, controller2, controller3, controller4] and color

is one of [white steel red orange gold olive green sage cyan teal cobalt blue violet purple pink crimson brown tan]

- **controller_set_player_character_type** [controller] [character]— Sets player character type for specified controller, where controller is one of [controller1, controller2, controller3, controller4] and character is one of [mp_masterchief mp_elite]
- **controller_set_emblem_info** [controller] [foreground_emblem] [background_emblem]— sets emblem for specified controller, where controller is one of [controller1, controller2, controller3, controller4] and foreground_emblem is some number in the range of [0, 31] inclusive and background_emblem is some number in the range of [0, 31] inclusive
- **controller_set_voice_output_setting** [controller] [setting]— Sets voice output setting for specified controller, where controller is one of [controller1, controller2, controller3, controller4] and setting is one of [default speakers headset none]
- **controller_set_voice_mask** [controller] [voice_mask]— Sets voice mask for specified controller, where controller is one of [controller1, controller2, controller3, controller4] and voice_mask is one of [none anonymous]
- **controller_set_subtitle_setting** [controller] [setting]— Sets subtitle setting for specified controller, where controller is one of [controller1, controller2, controller3, controller4] and setting is one of [automatic enabled disabled]

Misc

11/23/2022 • 2 minutes to read

Bitmap Curve Mode

Information on a setting which can allow your bitmaps to render up to twice as fast in the game: Bitmap Curve Mode.

Cheats.txt

Information on how to build a Cheats.txt file.

Console Commands

Information on Console Commands.

Gravity Lifts

Information on Gravity Lifts.

Optimal Texture Size Reference

Reference for use while creating .bitmap tags.

Tool Bitmap Suffix Reference

A list of all of the suffixes on .TIF files that Tool will automatically convert to specific types of Bitmap tags.

Bitmap Curve

11/23/2022 • 2 minutes to read

In each .bitmap tag is a setting which can allow your bitmaps to render up to twice as fast in the game: Bitmap Curve Mode. It's not a freebie though— the fast mode can look worse if the texture is very dark or very smooth. For the most part, however, they will be indistinguishable. The bitmap importer will automatically choose the fast mode if your bitmap is bright and will revert to the slow high-quality mode if the bitmap is dark.

You can manually control the bitmap curve setting within your .bitmap tag (see Figure 1)

- **Choose Best**: The importer chooses for you based on how bright your bitmap is.
- **Choose Fast**: Always uses the fast (but possibly ugly) mode.
- **Choose Pretty**: Always uses the slower, high-quality, mode.

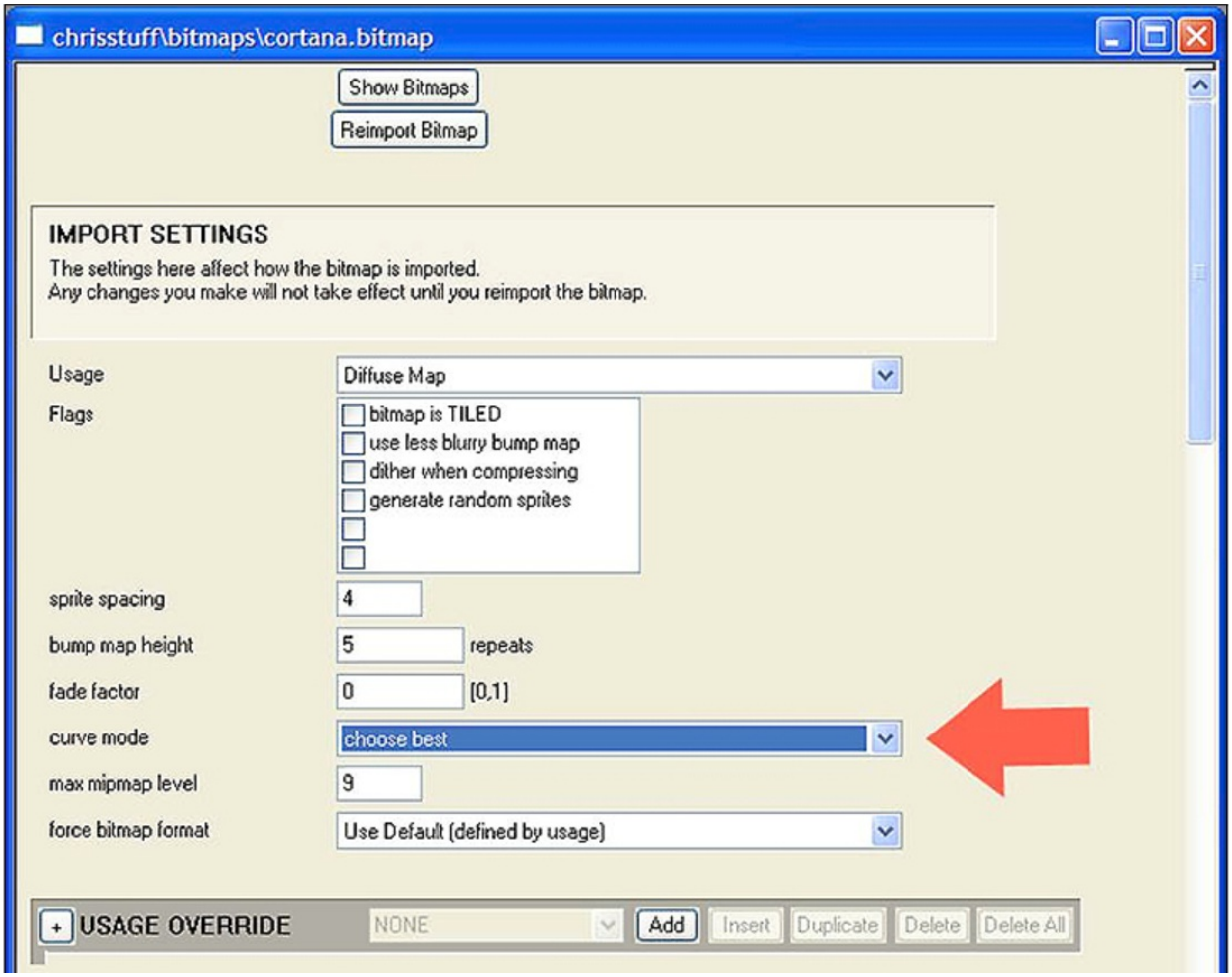


Figure 1 - .bitmap tag and the Bitmap Curve Mode setting.

> **NOTE**
>
> You need to re-import your bitmap after adjusting this setting for the changes to take effect!

Cheats.txt

11/23/2022 • 2 minutes to read

Add the following command to your init.txt file: **cheat_controller 1** (or whichever controller port you use).

Once that line is added to your init.txt, you can use hs-command shortcuts specified in your cheats.txt file.

Each button has a corresponding line, what appears before the r;; becomes the command. If cheat_controller is on, you can hold down the **back** button and hit any other button to activate that button's hs command.

In addition to the obvious things like game_speed 0 or error_geometry_hide_all, this can be useful for things like toggling oft-used commands such as:

```
(set debug_structure_automatic (not debug_structure_automatic)); x-button
```

The order of the lines is intentional and immutable— you can't assign commands to the start or back button.

A Sample Cheats.txt file:

```
; left trigger
; right trigger
; dpad-up
; dpad-down
; dpad-left
; dpad-right
; start (NOT USABLE)
; back (NOT USABLE)
; left-thumb
; right-thumb
game_speed 1; a-button
game_speed 4; b-button
(set debug_structure_automatic (not debug_structure_automatic)); x-button
; y-button
; left bumper
; right bumper
```

Console Commands

11/23/2022 • 2 minutes to read

Console commands allow you to access the game via a command prompt, allowing you to view and set scripts and functions.

There are four types of console commands:

- **hs_function** — defines behavior, such as object_set_permutation
- **hs_global** — defines data, such as debug_objects
- **scripts** < from .hsc file
- **globals** < from .hsc file

Entering Console Commands

**** Access the Console****

Press tilde (~) key to access the in-game command prompt. Use ~ for Sapien. This will allow you to type in console commands.

Autocomplete Commands

You can autocomplete any string you type in by just hitting the tab key. This will list all of the commands that start with the given string.

For example, if you type sa and press tab, you will see Figure 1.

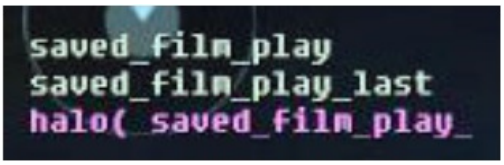


Figure 1 - Autocompleted Command.

Find

Type find and any string will list all of the commands that include the given string.

For example, if you type find physic, you will see Figure 2.

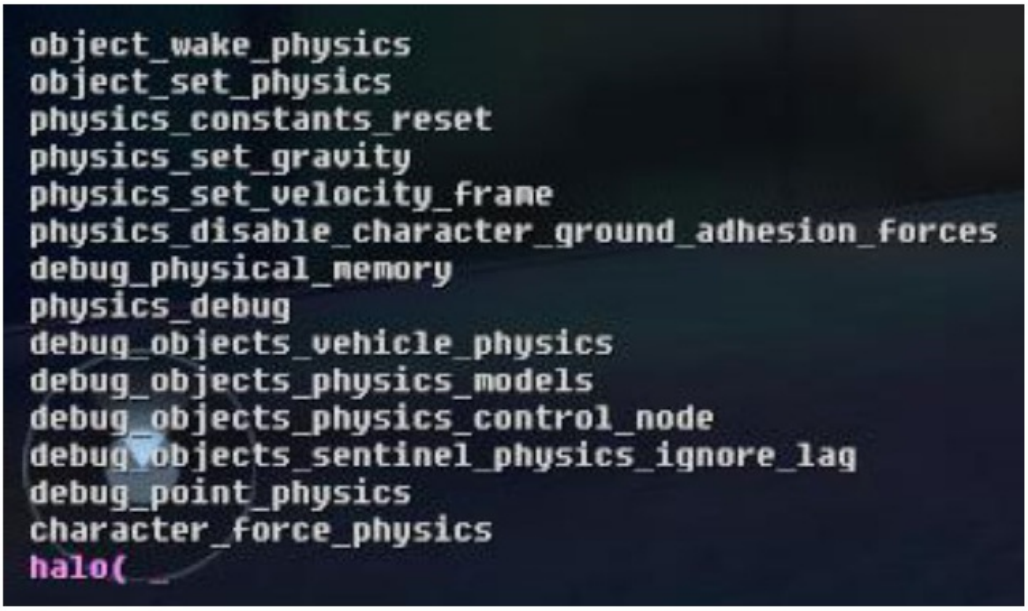


Figure 2 - Find Command Output.

Help

Type in help and a console command will list the command and the required parameters plus a description of the console command. Find is available for hs_functions only— other command types are not supported.

For example, if you type help object_set_permutation, you will see Figure 3:



Figure 3 - Help Command Output.

Debug Menu

The most common console commands have been wrapped in a GUI known as the Debug Menu. Press the back and start keys simultaneously to bring up the debug menu.

> **NOTE**
> Press the back key first then hold it then press the start key to avoid bringing up the game ui.

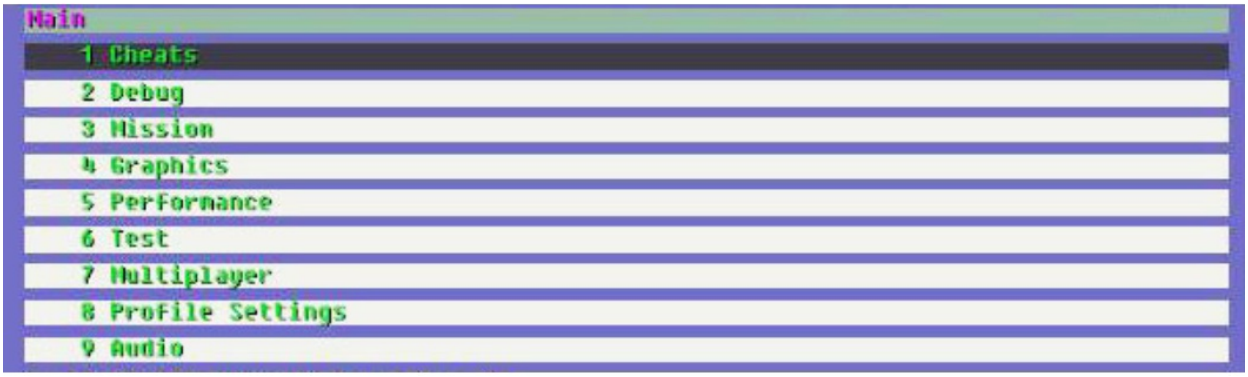


Figure 4 - The Debug Menu.

Creating Text List of All Halo Script Commands (script_doc)

You can export a list of the current hs_functions by typing in script_doc on the PC console (inside Sapien). This will create \halo3\main\hs_doc.txt.

Example:

```
(object_set_permutation \<object> \<string_id> \<string_id>)
```

Gravity Lifts

11/23/2022 • 2 minutes to read

To anchor a crate with working renders and physics models or to make the physics capsule push in a direction, you need to create a phantom volume in your physics tag. Do the following:

1. Go to your physics tag and r;Add a phantom volume (3rd field down). In this phantom volume you identify the appropriate marker (which is linked to the model).
2. Set the velocity and acceleration. Look at other lift's physics tags to determine the axis and directional velocity and acceleration to get a good start on appropriate settings.
3. Go to the materials area and link the phantom volume to the respective material.

> **NOTE**
> You need to have a unique material per phantom volume. Thus, if your lift is composed of three separate phantom volumes, you need three separate materials.

Optimal Texture Size Reference

11/23/2022 • 2 minutes to read

The following is intended as a reference for use while creating .bitmap tags.

Compressed Textures

- **Supported Sizes**: Any multiples of 4
- **Optimal Sizes**: Multiples of 128 (128, 256, 384, 512, 640, 768, 896, 1024, ...)

> **WARNING**
> As soon as you go over a multiple of 128, you use the same memory as the next higher multiple of 128!

So, a 132x260 texture uses the same amount of memory as a 256x384 texture. Also, the small mip map levels waste memory so, if possible, use a few higher-rez textures instead of many low-rez textures.

Again, it is much better to use a single large texture than to use multiple small ones that add up to the same size. For example, four compressed textures that are 128x128 use twice as much memory as a single 256x256 compressed texture. Refer to the memory efficiency chart below for details:

Memory Efficiency for Compressed Textures

BITMAP SIZES	MEMORY EFFICIENCY
16	(2.08%)
32	(4.17%)
64	(11.11%)
128	(33.33%)
256	(66.66%)
384	(70.59%)
512	(88.89%)
640	(68.03%)
768	(79.99%)
896	(89.50%)
1024	(96.97%)

Uncompressed Textures

- **Supported Sizes**: Any
- **Optimal Sizes**: multiples of 32 (32, 64, 96, 128, 160, 192, 224, 256, 288, ...)

> **WARNING**
> As soon as you go over a multiple of 32, you use the same memory as the next higher multiple of 32!

Also, for 32x32 and smaller mip levels, 2-channel and 1-channel textures use the same memory as 4-channel textures. So, at low resolutions, you don't gain anything by using fewer channels.

Tool Bitmap Suffix Reference

11/23/2022 • 2 minutes to read

The table below lists all of the suffixes on .TIF files that Tool will automatically convert to specific types of Bitmap tags.

BITMAP NAME SUFFIX	BITMAP TAG TYPE
_3d	3D texture
_blend	blend map for terrain
_bump	bump map (form height map)
_cc	change color map
_cube	cube map (reflection map)
_detail	detail map
_detailbump	detail bump map (uses fade factor like illum and detail maps)
_diff	diffuse map
_diffuse	diffuse map
_dsprite	sprite (double multiply, gray background)
_float	float map (very big)
_height	height map for parallax
_illum	self-illumination map
_lactxl	luminance/alpha map (4-bit)— particles only
_ladxn	luminance/alpha map (8-bit)— particles only
_msprite	sprite (blend, white background)
_spec	specular map
_sprite	sprite (additive, black background)
_ui	interface bitmap
_vec	vector map
_warp	warp map (EMBM)

BITMAP NAME SUFFIX	BITMAP TAG TYPE
_zbump	Zbrush bump map (from bump map)

Halo 3 ODST Mod Documentation Coming Soon

11/23/2022 • 2 minutes to read

Halo Reach FBX to GR2

11/23/2022 • 52 minutes to read

Overview

This manual covers conversion from the **.fbx** format to the **.gr2** format that needs to be done for custom models and levels when adding them to Halo Reach. And, also, it covers the import of the resulting **.gr2** file to tag files used by the game.

FBX is treated here as an initial format. I.e., it is assumed that you create your custom model or level in 3D editing software – Autodesk 3ds Max, Autodesk Maya, or Blender – and then export it to the FBX file. And, after that, use this FBX file as the source file for your model or level in Halo Reach.

GR2 is an intermediate format. You will use your GR2 files to convert them to the regular tag files (.model, scenario, etc.). This can be done using the **import** command of **tool.exe** or **tool_fast.exe**, with the help of an additional sidecar file (**<name_of_tag>.sidecar.xml**).

Conversion from FBX to GR2 is performed via the **tool.exe** or **tool_fast.exe** apps, using the **fbx-to-gr2** command. Along with an initial FBX file, this conversion also needs an auxiliary file in the **JSON** format, which will contain the meta-information about the model or level that is absent in the initial FBX.

After converting FBX to GR2 and GR2 to tag files, you will be able to use these tag files in regular Halo Reach modding pipelines.

FBX to GR2 Conversion

The "fbx-to-gr2" Command

Conversion is performed by the **fbx-to-gr2** command of **tool.exe** or **tool_fast.exe**.

The syntax of this command is the following:

```
tool_fast.exe fbx-to-gr2 <fbx-model> <json-file> <gr2-model>
```

Where:

- **<fbx-model>** – The path to the source FBX file, including its name. You can specify here either the full path to the FBX file or a relative path.
- **<json-file>** – The path to the auxiliary JSON file, including its name. This file is necessary to provide the meta-information about the model or level that is absent in the initial FBX. The value of this parameter is required for the correct conversion; *however, the existence of the file itself is not required*. I.e., at the first launch, you can specify the path to the *non-existing file* – to the location where you want to create it – and the system *will create a valid JSON file at this location automatically*, will fill it with some valid data, and will perform conversion with this (default) info. After that, you can modify this file and add necessary additional info to it, to receive a more "exact" GR2 file as a result of next conversions. For details, see JSON File below.
- **<gr2-model>** – The path to the output GR2 file, including its name.

For example:

```
tool_fast.exe fbx-to-gr2 "C:\Program Files
(x86)\Steam\steamapps\common\HREK\data\levels\example_level\example_level.fbx" "C:\Program Files
(x86)\Steam\steamapps\common\HREK\data\levels\example_level\example_level.json" "C:\Program Files
(x86)\Steam\steamapps\common\HREK\data\levels\example_level\example_level.gr2"
```

> **NOTE**
>
> If your paths contain folders with spaces inside names, you need to put these paths in the double-quotes when executing the **fbx-to-gr2** command.

Or, you can use relative paths, if you execute this command from the root folder of Halo Reach Editing Kit (HREK):

```
tool_fast.exe fbx-to-gr2 data\levels\example_level\example_level.fbx
data\levels\example_level\example_level.json data\levels\example_level\example_level.gr2
```

Or, you can use **tool.exe** instead of **tool_fast.exe** and some temporary folder outside HREK:

```
tool.exe fbx-to-gr2 "C:\temp folder\example_level.fbx" "C:\temp folder\example_level.json" "C:\temp
folder\example_level.gr2"
```

Requirements for Rigged 3D Models

There is a specific requirement for rigged 3D models (models with a skeleton made from bones; these models have an Armature in Blender).

Particularly, when creating these models, you need to ***store all transforms in the nodes of the skeleton***, not in their meshes. See the FBX file in the example_tentacle sample.

JSON File

The **fbx-to-gr2** command requires a JSON file for successful conversion. This file is necessary to provide the meta-information about the model or level that is absent in the initial FBX, but still required by the game engine.

For example, if your FBX file contains some meshes and you want to generate correct tag files based on it, these tag files need to contain information on the *types* of these meshes from the point of view of game engine. E.g., you need to mark water surfaces as water surfaces, portals as portals, and so on. Otherwise, your model or level may appear or behave in the engine incorrectly.

However, at the first conversion, the initial JSON file is created by **tool.exe** or **tool_fast.exe** automatically. All you need to specify at the first conversion is the preferable location of this JSON that will be created.

If you do, the system will create it for you and will fill it with the *default* values of the properties of objects in your FBX. For example, it will use "**_connected_geometry_mesh_type_default;**" as the type for all meshes. And, after that, the system will use it for conversion, so you will still get a target GR2 file (based on these default values).

After that, to obtain a better GR2, you can tune the properties specified in the JSON file, assign the necessary values to them, and reconvert your GR2 file using the modified JSON file and the same FBX file.

Modifications to the JSON file can be done in any textual editor. For example, to better handle the JSON formatting and avoid syntax errors, you can use Visual Studio Code editor, available freely at https://code.visualstudio.com

The structure of this JSON file is the following:

- **nodes_properties** – this section contains the list of all nodes from the FBX file. (Roughly speaking, a "node" here is a single item that can be selected in the **Outliner** of a 3D Editing Tool and that may correspond to a bone, marker, mesh, and so on.) Every node in this list has a set of properties that you can modify. For example:



Fig 1. View of the node properties for an FBX file.

- **meshes_properties** – this section contains the list of all meshes from the FBX file. Similarly to the list of nodes, every mesh in this list has a set of properties that you can modify.
- **material_properties** – this section contains the list of materials from the FBX file used by your model and allows you to specify properties of these materials. Particularly, here you can define the shader tag files that correspond to every material. For details, see Material Properties.

> **NOTE**
> If your model contains Regions or you want to specify some overrides of global materials, you will also need to add the **string_table** section to the JSON file. In this case the order of sections will be the following: **string_table**, **nodes_properties**, **mesh_properties** and **material_properties**.

Names of all possible properties, types (format) of their values, and, for some properties, the list/range/format of their possible values are listed in the Appendix B: JSON Parameters.

Most of these properties and values have self-explaining names, e.g. the **mesh_type** property can have the **_connected_geometry_mesh_type_collision** value, which specifies that this mesh needs to be treated as a collision mesh by the game engine.

> **NOTE**
> Most of these fields need to be specified with the "**bungie_**" prefix in JSON file, please note that this prefix is omitted in the names of properties in Appendix B.

NOTE

Some parameters used in the samples (e.g. "**bungie_is_cinematic_object**", "**halo_export**", etc.) are currently not used in the game engine and are legacy parameters. They are used in samples, since they are used in Halo 4 and Halo 2 Anniversary. However, they are not used in Halo Reach and are not listed in Appendix B.

NOTE

In the future, we will generate and publish a pack of original JSON files for all in-game models that are used in the Halo Reach title. This will probably help modders to modify them, since there will be an original JSON that can be used as a base.

NOTE

You can use JSON files from the provided samples as a basis for your own JSON files, see Appendix A: Samples.

Some properties (**object_type**, **object_ID**, **object_animates**) are the main properties of an object and are used for both for node objects in the "**nodes_properties**" section and, separately, for mesh objects in "**meshes_properties**".

Most properties in the Appendix B can be specified for objects of particular type only. The type of object is specified for every object in the **object_type** property. The type of the object that the property relates to typically can be identified by the suffix in its name. E.g, properties with the "**mesh**" suffix typically correspond to the objects with **object_type** equal to **_connected_geometry_object_type_mesh**. Along with that, you can find the correct object type for every property in the Appendix B, in the **Additional Info** column.

NOTE

Moreover, due to the legacy components of the system, there is an additional requirement for nodes that are meshes (i.e. **_connected_geometry_object_type_mesh** as the type of the node), if they are listed in both **nodes_properties** and **meshes_properties**. In this case, properties with the "**mesh**" suffix need to be duplicated in both descriptions of an object (both in **nodes_properties** and **meshes_properties**). However, this applies only to objects that are defined as meshes in both these sections. If, in fact, there are two different objects (e.g. a frame in **nodes_properties** that contains a mesh in **meshes_properties**), this is not necessary.

For example, using the list of properties in Appendix B you can add a Hinge constraint with 180° and -180° angle limits between two nodes of a FBX file. If you want to do this with the two nodes (e.g. **Parent_Node** and **Child_Node**), you will need to modify their properties in the **nodes_properties** section as follows:

```
{
    "nodes_properties": {
        ...,
        "Parent_Node": {
            ...,
            "bungie_physics_constraint_child": "Child_Node",
            "bungie_physics_constraint_hinge_min": "-180",
            "bungie_physics_constraint_hinge_max": "180"
        },
        "Child_Node": {
            ...,
            "bungie_physics_constraint_parent": "Parent_Node"
        }
    },
    ...
}
```

However, since usage of constraints is possible for physical objects only, this code needs to be specified in the JSON file for the physical model (e.g. typically a separate FBX that needs to be also converted to GR2 and then to the **.physics_model** tag file). And, along with the code above, you will also need to modify the **object_type** of these nodes. The physics engine in Halo Reach supports two types of physical objects:

- A physical mesh, i.e., the **_connected_geometry_mesh_type_physics** type of the mesh.
- A marker with the physics constraint, i.e. either **_connected_geometry_marker_type_physics_hinge_constraint** or **_connected_geometry_marker_type_physics_socket_constraint** types of markers. For the specified parameters, we can use markers, so the final code will be similar to the following:

```
{
    "nodes_properties": {
        ...,
        "Parent_Node": {
            "bungie_object_type": "_connected_geometry_object_type_marker",
            "bungie_marker_type": "_connected_geometry_marker_type_physics_hinge_constraint",
            ...,
            "bungie_physics_constraint_child": "Child_Node",
            "bungie_physics_constraint_hinge_min": "-180",
            "bungie_physics_constraint_hinge_max": "180"
        },
        "Child_Node": {
            "bungie_object_type": "_connected_geometry_object_type_marker",
            "bungie_marker_type": "_connected_geometry_marker_type_physics_hinge_constraint",


            ...,
            "bungie_physics_constraint_parent": "Parent_Node"
        }
    },
    ...
}
```

Regions and Global Material Overrides. String Table ("string_table") in JSON

If your model contains Regions or you want to specify some overrides of global materials, your JSON file will need to be a bit different.

NOTE

A Region is a homogeneous and somewhat independent section of a model. Regions are set up so that different areas of the same model can have different properties. For example, the head of a marine is one region of the model, while the body is another. Creating regions allows us to have variations between parts of the model (without having to load or create separate models). Each region can have different permutations and states than other regions of the model.

NOTE

Not all models need to be split into Regions. If your model does not need it (and you do not want to make overrides to global materials), you can skip this section. It is totally optional.

If your model needs Regions:

Particularly, in the **meshes_properties** section of JSON, you will need to mark all meshes of your model that correspond to regions with the face_region property and specify the name of the corresponding region as a value of this property. The syntax is the following:

```
"bungie_face_region": "<name_of_region>"
```

For all meshes that are not marked with the **face_region** property, the system will assume that they correspond to the "**default**" region, which exists for every model.

For example, if you want your model to have two separate regions ("**body**" and "**head**") along with the "**default**" region, your **meshes_properties** section in JSON may look like the following:

```
"meshes_properties": {
    ...,
    "Cube1": {
        "bungie_object_type": "_connected_geometry_object_type_mesh",
        ..., // there is no "bungie_face_region" property for this mesh, so it will be
        ... // automatically assigned to "default" region, which always exists for a model.
    },
    "Cube2": {
        "bungie_object_type": "_connected_geometry_object_type_mesh",
        ...,
        "bungie_face_region": "body",
        ...
    },
    "Cube3": {
        "bungie_object_type": "_connected_geometry_object_type_mesh",
        ...,
        "bungie_face_region": "head",
        ...
    },
    ...,

}
```

But that's not all. Along with that, you will need to specify a ***string table*** for your regions. The string table is the mapping (key-value) table between names and indices. It is necessary, since, regardless of the fact that most of the tag files use names, indices corresponding to these names are actually passed to the game engine and processed by it (to determine what to render). Along with that, some tag files also explicitly use indices.

NOTE

Indices also define the order of the particular region in the list of regions in various tag files opened in Foundation. I.e., you can identify the index (starting with "**0**") of the particular region by its position in these lists. Needless to say, if you are importing a new model that does not exist in the game, you may define indices (and order of regions) as you wish. However, when you create a substitution for an existing model, you will need to define indices of regions exactly as they are defined in the original model.

This string table can be defined either in the JSON file or in the sidecar XML file. However, if you specify its values both in JSON and sidecar XML, the sidecar XML version will have more priority and its values will overwrite the values of the JSON version. Overwriting will be performed by the comparison of names, if they match, the system will overwrite corresponding indices. For clarity, we recommend you to define this table either in sidecar XML, or in JSON.

If you want to define the string table in JSON, you will need to add the supplemental **string_table** section to your JSON file, before the **nodes_properties** and **meshes_properties** sections.

Then, in this **string_table** section, you will need to add two properties:

- **regions_names** – the collection of names of regions. This collection must contain the default region (named "**default**") and names of all other regions that you used when you were marking meshes of regions with **face_region** property (see above). Except the "**default**" region, names within this collection and names defined by **face_region** property must be exactly the same.
- **regions_values** – the collection of indices that correspond to these names of region.

For example, if you want to define indices for regions used in the previous example, your string table may be defined like this:

```
{
    "string_table": {
        "regions_names": ["default", "head", "body"],
        "regions_values": ["0", "1", "2"]
    },
    "nodes_properties": {
        ...
    },
    "meshes_properties": {
        ...
    }
}
```

NOTE

Values of indices start with **0**. The "**default**" region is mandatory.

If necessary, the values of indices assigned to region names may be overwritten in the sidecar XML, using the corresponding <FaceCollection> tag containing a set of <FaceCollectionEntry> tags with Index and Name values, see below for details.

If your model needs Global Material Overrides:

If your model needs Global Material Overrides, you can define them in the String Table too. The mechanics here is the same, you add two properties to the **string_table** section:

- **global_materials_names** – the collection of names of global materials.0

- **global_materials_values** – the collection of indices corresponding to specified global materials.

For example:

```
{
    "string_table": {
        "global_materials_names": ["skin", "hair"],
        "global_materials_values": ["0", "1"],
        "regions_names": ["default", "head", "body"],
        "regions_values": ["0", "1", "2"]
    },
    "nodes_properties": {
        ...
    },
    "meshes_properties": {
        ...
    }
}
```

However, in this case, there is one important nuance: global materials cannot be defined in sidecar XML. You can specify an absolutely correct <FaceCollection> tag for them there (our examples contain this code), but all global material values specified in sidecar XML will be ignored during importing due to the legacy code. Thus, global materials overrides can be specified in JSON only.

Material Properties (the "material_properties" section)

If you want to correctly set up the materials used by your model, you need to add one more section to the JSON file.

Particularly, you will need to add the "**material_properties**" section on the same level as "**nodes_properties**" and "**meshes_properties**"and after them.

This section should contain entries corresponding to all materials from the FBX file that you want to use in Halo Reach for your model. These entries will allow you to specify properties of these materials, particularly, the type of shader that needs to be used for this material and the path to it.

For example:

```
{
    "nodes_properties": {
        ...
    },
    "meshes_properties": {
        ...
    },
    "material_properties": {
        "health_pack_rubber": {
            "bungie_shader_path": "objects\\equipment\\health_pack\\shaders\\health_pack_rubber",
            "bungie_shader_type": "shader"
        },
        "health_pack": {
            "bungie_shader_path": "objects\\equipment\\health_pack\\shaders\\health_pack",
            "bungie_shader_type": "shader"
        }
    }
}
```

The structure of every entry here is the following:



Fig 2. Structure of the material property entry.

- The name of the entry (**A** above) – this is the name of the material. It must be absolutely the same as the name of the material in the FBX file.

 For example, "**health_pack_rubber**".

 - **shader_type** – the value of this property (**B** above) is the type of the shader that will be used for this material. For shader files, this corresponds to the extension of the shader tag file.

 For example, "**shader**". Or, "**shader_water**".

 You can refer to the Appendix B: JSON Parameters for the full list of possible values of the **shader_type** property.

 - **shader_path** – the value of this property (C above) is the path to the shader tag file (of the specified **shader_type**). This path needs to be specified relative to the **tags** folder, without file extension, and with double backslashes as path delimiters.

 For example, "**objects\\equipment\\health_pack\\shaders\\health_pack_rubber**"

As you can see, the general use of the "**material_properties**" section is to map materials from the FBX file to appropriate shader tag files.

> **WARNING**
> If you miss some materials from FBX or will not add the "**material_properties**" section at all, the system will assign the default invalid shader (**tags\shaders\invalid.shader**) to all materials from FBX that were not specified in JSON.

However, there is also a special case. Using a material entry in the "**material_properties**" section, you can also override properties of the mesh that this material is assigned to, in the area of the mesh covered with this material.

> **NOTE**
> The case described below is a workaround that is intended for special situations. Please, use it accurately and avoid it when possible. We recommend you to define the mesh properties in a typical way, without such overriding.

If you specify "**override**" as a value of **shader_type**, you will be able to assign a special value to **shader_path** property. Particularly, you will be able to specify a new value of the particular mesh property that will override its initial value in this area

For example, if you want to set up a particular part of the mesh surface as a sky in Halo Reach and, in your FBX, this part of the surface is covered with the material named "sky_material", you can specify the following in the properties of this material in the JSON file:

```
    "sky_material": {
        "bungie_shader_type": "override",
        "bungie_shader_path": "bungie_face_type=_connected_geometry_face_type_sky"
    }
```

Where "**bungie_face_type=_connected_geometry_face_type_sky**" value of **shader_path** will set **face_type** property *of this part of the surface* of this mesh to "**_connected_geometry_face_type_sky**", which means that it will be treated as sky by Halo Reach.

The syntax here is rather simple:

```
"bungie_shader_path": "<name_of_mesh_property>=<value_of_this_property>"
```

Please note that syntax does not support delimiters between properties and, because of that, you cannot override more than one mesh property by a single material.

However, in a rare case when it is necessary to override multiple mesh properties for the particular part of the mesh at once, you can create multiple materials in 3d editing software, cover the same part of the mesh surface with them, and add all of these materials to the "**material_properties**" section, overriding every necessary mesh property on a per material basis.

Compatibility with FBX Files Made for Previous Versions of the Game

In general, FBX files made for previous Halo titles are compatible with the FBX-to-GR2 conversion and can be imported to Halo Reach.

However, the limitation for rigged 3D models (see above) should also be taken into account

Using GR2 for Importing Tags

To import a GR2 file to the game engine format (set of tag files), you need to create a sidecar XML file for it and use it during import.

Creation of Sidecar XML file

To create a sidecar XML file for your GR2 file, we recommend you to use samples provided with this manual – see Appendix A: Samples.

Particularly, you can take one of the sidecar XML files from these samples, rename it to **<name_of_your_model>.sidecar.xml**, modify some fields in it to match your GR2 file, and use it while importing. The process of these modifications is briefly described below.

> **NOTE**
> The full process of the creation of a sidecar XML from scratch is a separate topic and is out of the scope of this guide.

> **NOTE**
> The main idea of the sidecar XML in the current pipeline is ***not*** in providing the *full* description of a model/level. Its main idea here is to provide a ***minimum*** description of the model/level that is required for the successful import. After the successful import, all other necessary properties of the model/level tag files can be set in Foundation.

Typically, the process of adaptation of existing sidecar XML to your needs is the following:

- Browse samples and, for simplicity, locate the sample that is closer to the type of the 3d model you need to import. For example, choose **example_level.sidecar.xml**, if you have a level. Or, choose

example_tentacle.sidecar.xml, if you have a model with multiple nodes. Or, choose **example_health_pack.sidecar.xml**, if you have a model with custom textures.

> **NOTE**
> If necessary, you can combine samples as you need, by copy-pasting and changing necessary parts of sample sidecar XML files.

- Rename selected sample sidecar XML into **<name_of_your_model>.sidecar.xml**. The name of this file affects the import process. It must be the same as the name of the model folder (same as the model name).
- For your convenience and to match default paths to folders in the sample sidecar XML files, organize the folder structure for your files:
 - Create a folder for your model/level somewhere within the **data** folder of the HREK directory. For example, **data\levels\custom_levels\example_level**
 - Put your sidecar XML into this folder. E.g. **data\levels\custom_levels\example_level\example_level.sidecar.xml**.
 - In this folder, create two subfolders: **models** – for the source of your model, and the **export** folder with a **models** subfolder –for its GR2 file.
 - Put your FBX and JSON files into the **models** directory and GR2 file into **export\models**.
 - If you have image files and textures that are used by your model, put them into the **bitmaps** folder.
- In your sidecar XML file:
 - Do not modify the description of the sidecar XML file in the <Header> section. You can modify it, but we do not recommend that.

```
<Header>
    <MainRev>0</MainRev>
    <PointRev>6</PointRev>
    <Description>created by SidecarFile 0.6</Description>
    <Created>4/16/2009</Created>
    <By>jrodgers</By>
    <DirectoryType>TAE.Shared.NWOAssetDirectory</DirectoryType>
    <Schema>1</Schema>
</Header
```

 - In the <Asset> section, specify the name of your object to be imported, its type, and specify the output tag files that need to be generated as a result of import. For details, see <Asset> tag and <OutputTagCollection> and <OutputTag> tags below.

```
<Asset Name="example_level" Type="scenario">
    <OutputTagCollection>
        <OutputTag
Type="scenario_lightmap">levels\example_level\example_level_faux_lightmap</OutputTag>
        <OutputTag Type="structure_seams">levels\example_level\example_level</OutputTag>
        <OutputTag Type="scenario">levels\example_level\example_level</OutputTag>
    </OutputTagCollection>
    <SharedAssetCollection />
</Asset
```

 - If necessary, in the <Folders> section, modify the folders that will be used during import for various

data of the imported object, see <Folders> tag for details. For a folder of every type, the path is specified relative to the folder where the sidecar XML is located.

> **NOTE**
>
> Default paths to folders that are specified in the sample files assume that the source files of your model are located in **models**, **export\models**, and **bitmaps** folders, as specified above. Typically, if your files are located as described in previous steps, you do not need to modify anything in the <Folders> section. We recommend you to organize files this way and do not change the <Folders> section.

> **NOTE**
>
> The set of child-tags (particular folders) may be different here, depending on the type of the model (whether it is a model or a level).

> **NOTE**
>
> Not all of these folders will be used during import, since some data may be not necessary for your model. However, we recommend you to do not remove unnecessary folders since they may be necessary for import of other models, so it is a good idea to have a list of all of them somewhere.

```
<Folders>
    <Reference>\reference</Reference>
    <Temp>\temp</Temp>
    <SourceModels>\models\work</SourceModels>
    <GameModels>\models</GameModels>
    <GamePhysicsModels>\models</GamePhysicsModels>
    <GameCollisionModels>\models</GameCollisionModels>
    <ExportModels>\export\models</ExportModels>
    <ExportPhysicsModels>\export\models</ExportPhysicsModels>
    <ExportCollisionModels>\export\models</ExportCollisionModels>
    <SourceAnimations>\animations\work</SourceAnimations>
    <AnimationRigs>\animations\rigs</AnimationRigs>
    <AnimationTransplants>\animations\transplants</AnimationTransplants>
    <GameAnimations>\animations</GameAnimations>
    <ExportAnimations>\export\animations</ExportAnimations>
    <SourceBitmaps>\bitmaps</SourceBitmaps>
    <GameBitmaps>\bitmaps</GameBitmaps>
    <CinemaSource>\cinematics</CinemaSource>
    <CinemaExport>\export\cinematics</CinemaExport>
    <ExportBSPs>\export\models</ExportBSPs>
    <GameBSPs>\models</GameBSPs>
    <SourceBSPs>\models\work</SourceBSPs>
    <RigFlags>\flags</RigFlags>
    <RigPoses>\poses</RigPoses>
</Folders>
```

- If necessary, in the <FaceCollections> section, specify the **String Table for regions**. If you have already done this in JSON, the values specified in sidecar XML will overwrite the JSON version. See <FaceCollections>, <FaceCollection>, <FaceCollectionEntries>, and <FaceCollectionEntry> tags for details

```
<FaceCollections>
    <FaceCollection Name="regions" StringTable="connected_geometry_regions_table" Description="Model
regions">
        <FaceCollectionEntries>
            <FaceCollectionEntry Index="0" Name="default" Active="true" />
        </FaceCollectionEntries>
    </FaceCollection>
</FaceCollections>
```

- In the <Contents> section, define the content of the main tag files (listed in the <Asset> section) we want to create. And, also, specify what additional tag files need to be created and linked to the fields of main tag files. For details, see <Contents>, <Content>, <ContentObject>, <ContentNetwork>, <InputFile>, and <IntermediateFile> tags below.

> **NOTE**
>
> Textures for your models need to be defined in the source FBX file of the model, with names corresponding to the existing/to-be-created **.shader** tag files. These **.shader** tag files are created in Foundation (separately from the import process). See Importing Materials (Shaders) for details.

- Save your sidecar XML file and proceed to importing GR2 to tag files, as described in Import to Tags.

<Asset> tag

The <Asset> tag sets the main object to be imported and, via child-tags (see below), the *main* output tag files that need to generated as a result of the import. It may contain the <OutputTagCollection> tag and <SharedAssetCollection> tag (tag for the list of shared assets, not covered by this guide).

> **NOTE**
>
> Additional output tag files can also be created when you will be specifying contents of the main files using the children of the <Contents> tag, see below.

For example:

```
<Asset Name="example_health_pack" Type="model">
    <OutputTagCollection>
        <OutputTag Type="model">objects\equipment\example_health_pack\example_health_pack</OutputTag>
        <OutputTag Type="crate">objects\equipment\example_health_pack\example_health_pack</OutputTag>
    </OutputTagCollection>
    <SharedAssetCollection />
</Asset>
```

Attributes:

- Name – the name of the asset to be imported. Must be the same as the name of the tag file(s).
- Type – the type of the asset to be imported. The type specified in this field does affect the import process. For example, in our samples the following asset types are used:
 - "model" – corresponds to models of objects (not levels).
 - "scenario" – corresponds to models of levels.

<OutputTagCollection> and <OutputTag> tags

The <OutputTagCollection> tag groups the <OutputTag> tags and is used as a container for them. This tag can be used within the <Asset> and <ContentObject> tags and allows you to set the tag files that will be generated for this asset or content object as a result of import.

For example:

```
<Asset Name="example_level" Type="scenario">
    <OutputTagCollection>
        <OutputTag Type="scenario_lightmap">levels\example_level\example_level_faux_lightmap</OutputTag>
        <OutputTag Type="structure_seams">levels\example_level\example_level</OutputTag>
    <OutputTag Type="scenario">levels\example_level\example_level</OutputTag>
    </OutputTagCollection>
    <SharedAssetCollection />
</Asset>
```

The <OutputTag> tag specifies the particular tag file that needs to be generated during the import for an asset or content object. This tag can be used within the <OutputTagCollection> tag.

For example:

```
<OutputTagCollection>
    <OutputTag Type="scenario_lightmap">levels\example_level\example_level_faux_lightmap</OutputTag>
    ...
</OutputTagCollection
```

Attributes:

- Type – the type of the tag file to be generated, which corresponds to the file extension of this tag file. For example, the Type="scenario_lightmap" will generate the **.scenario_lightmap** tag file. The name of this file is set by the value of this tag. (However, lightmaps for levels need to be additionally calculated after the import, see below).

Value:

- The value of this tag sets the path to tag file to be generated and its name. This path is set relative to **tags** folder of the HREK directory. The name of the tag file is set in the end of this path. For example, the levels\example_level\example_level_faux_lightmap value, in combination with Type="scenario_lightmap" will create the **example_level_faux_lightmap.scenario_lightmap** in the **tags\levels\example_level** directory.

<Folders> tag

The <Folder> tag defines the set of folders that will be used during import for various data of the imported object. It contains multiple child-tags that correspond to data of theparticular type. Every such child-tag tells the system where it needs to look for data of the particular type.

For example, the <GameCollisionModels> child-tag tells the system where to look for the source files of the collision models (FBX and JSON) and the <ExportCollisionModels> child-tag – where to look for intermediate (converted) files of the collision models (GR2).

For a folder of every type, the path is specified relative to the folder where the sidecar XML is located.

The set of child-tags (particular folders) may be different here, depending on the type of the model (whether it is a model or a level).

For example, for a model:

```
<Folders>
    <Reference>\reference</Reference>
    <Temp>\temp</Temp>
    <SourceModels>\models\work</SourceModels>
    <GameModels>\models</GameModels>
    <GamePhysicsModels>\models</GamePhysicsModels>
    <GameCollisionModels>\models</GameCollisionModels>
    <ExportModels>\export\models</ExportModels>
    <ExportPhysicsModels>\export\models</ExportPhysicsModels>
    <ExportCollisionModels>\export\models</ExportCollisionModels>
    <SourceAnimations>\animations\work</SourceAnimations>
    <AnimationRigs>\animations\rigs</AnimationRigs>
    <GameAnimations>\animations</GameAnimations>
    <ExportAnimations>\export\animations</ExportAnimations>
    <SourceBitmaps>\bitmaps</SourceBitmaps>
    <GameBitmaps>\bitmaps</GameBitmaps>
    <CinemaSource>\cinematics</CinemaSource>
    <CinemaExport>\export\cinematics</CinemaExport>
    <ExportBSPs>\export\models</ExportBSPs>
    <GameBSPs>\models</GameBSPs>
    <SourceBSPs>\models\work</SourceBSPs>
    <RigFlags>\animations\rigs\flags</RigFlags>
    <RigPoses>\animations\rigs\poses</RigPoses>
    <RigRenders>\animations\rigs\render</RigRenders>
    <Scripts>\scripts</Scripts>
    <FacePoses>\animations\rigs\poses\face_poses</FacePoses>
    <CinematicOutsource>\outsource</CinematicOutsource>
</Folders>
```

<FaceCollections>, <FaceCollection>, <FaceCollectionEntries>, and <FaceCollectionEntry> tags

The <FaceCollections> tag allows you to specify where the system will need to find values for list properties and what particular values should be used from these lists.

However, in our case, this tag and its child-tags is used ***only to define Regions of the model***.

> **WARNING**
>
> In samples, you can also see that the same tag was used to specify overrides of the global materials, but currently thispart of the functionality does not work. You can specify them using the <FaceCollections> tag structure, but all specified values that are related to global materials will be ignored during import due to the legacy code. See Regions and Global Material Overrides. String Table ("string_table") in JSON above for details.

The hierarchy of the child-tags of <FaceCollections> tag is the following: <FaceCollections> tag can have multiple <FaceCollection> tags. Every <FaceCollection> tag is linked to a certain list of elements (specified in the StringTable attribute) and contains <FaceCollectionEntries> tag that is the list of items in this collection. The <FaceCollectionEntries> tag may contain multiple <FaceCollectionEntry> tags. Every <FaceCollectionEntry> tag corresponds to a single element from the StringTable list.

Attributes and Values:

<FaceCollection> – Attributes:

- Name – The name of the collection. In our case, you can use only the "regions" value to specify Regions of the model. Our samples contain collections with the "global materials override" value too, but values specified for these collections are ignored during import.
- StringTable – The key that defines the particular table with collection values (which defines the mapping between indices and the names of the regions). In our case, you can use only the "connected_geometry_regions_table" value that will tell the system that you are specifying Regions of the model. Samples contain collections with the "connected_geometry_global_material_table" value too, but

values specified for these collections are ignored during import.

> **NOTE**
>
> The StringTable is the mapping table between names and indices. Most tag files use names, but, actually, indices corresponding to these names are passed to the game engine and processed by it (to determine what to render). Along with that, some tag files also use indices. The StringTable can be defined by in sidecar XML, using the set of <FaceCollectionEntry> tags with Index and Name values, see below. However, along with that, it can also be defined in the JSON file, see Regions and Global Material Overrides. String Table ("string_table") in JSON above. We recommend you to define this table either in sidecar XML or in JSON. However, if you specify its values both in JSON and sidecar XML, the sidecar XML version will have more priority and its values will overwrite the values of the JSON version.

> **NOTE**
>
> If your model uses Regions, then, along with the creation of StringTable, you need to mark meshes of the FBX model that correspond to specific regions with the **face_region** property in the JSON file and define names of regions for these meshes using this property. See Regions and Global Material Overrides. String Table ("string_table") in JSON above for details.

- Description – The description of this field.

<FaceCollectionEntry> – Attributes:

- Index – The index of the element from the StringTable table. In our case, the index of the Region.
- Name – The name of the element from the StringTable table. In our case, the name of the Region.
- Active – Whether this value is active (enabled, "true") or inactive (disabled, "false").

> **NOTE**
>
> For example, in the sample below for "regions", you will add 3 regions with 3 corresponding indices to the list of the regions of the model ("connected_geometry_regions_table") and all these regions will be rendered for the model (since Active="true" for them).

For example

```
<FaceCollections>
    <FaceCollection Name="regions" StringTable="connected_geometry_regions_table" Description="Model
regions">
        <FaceCollectionEntries>
            <FaceCollectionEntry Index="0" Name="default" Active="true" />
            <FaceCollectionEntry Index="1" Name="head" Active="true" />
            <FaceCollectionEntry Index="2" Name="body" Active="true" />
        </FaceCollectionEntries>
    </FaceCollection>
</FaceCollections
```

<Contents>, <Content>, <ContentObject>, <ContentNetwork>, <InputFile>, and <IntermediateFile> tags

The <Contents> tag – via its child-tags (see below) – defines the content of the tag files we want to create. Moreover, along with that, when specifying content for the particular fields of the model/level, you are also able to create the additional tag files that need to be created and link these fields to these tags. For example, every **.model** tag file has a **render model** field that links to the **.render_model** tag, which in its turn contains the render model data. So, when specifying content for this **render model** field, you are actually setting the source (<InputFile> and <IntermediateFile>) for the render **.render_model** tag, which the system will also generate

based on this source (you specify it via <OutputTagCollection>).

The hierarchy of the <Contents> tag, with its child-tags, is the following:

```
<Contents>
   <Content> – corresponds to content of a particular main output tag file listed in the <Asset> tag.
       <ContentObject> – corresponds to content of a particular field in the main output tag file listed in
the <Asset> tag.
           <ContentNetwork> – corresponds to a particular variant (style) of a model. There may be multiple
variants of the model, with different input files. However, there always must be a "default variant.
               <InputFile/> – corresponds to the particular source file for this field and additional tag to
be generated, in our samples – the FBX file.
               <IntermediateFile/> – corresponds to the particular intermediate file for this field and
additional tag to be generated, in our samples – the GR2 file.
           </ContentNetwork>
           <OutputTagCollection> – (optional tag) corresponds to the set of additional output tag files to
be generated (based on the <ContentNetwork>) and, if necessary, linked to this field.
               <OutputTag/>
               <OutputTag/>
               ...
           </OutputTagCollection>
       </ContentObject>
       <ContentObject/>
       ...
   </Content>
</Contents>
```

Attributes and values:

<Content> – Attributes:

- Name – The name of the asset, listed in the <Asset> tag.
- Type – The type of the asset, listed in the <Asset> tag.

<ContentObject> – Attributes:

- Name – In our case, you can specify the blank value (""), which will tell the system to use the name of the model as Name.
- Type – The (type of the) field from this asset.

<ContentNetwork> – Attributes:

- Name – The name of the “style” (variant) of the model. There may be multiple variants of the model, with different Name values. However, there always must be a default variant where Name="default".
- Type – In our case, simply the blank value (""), to meet the requirements of the legacy format.

<InputFile> – Value:

- The path to the input file for the field and tag to be generated, relative to the **data** directory; in our samples – it’s the FBX file.

<IntermediateFile> – Value:

- The path to the intermediate file for the field and tag to be generated, relative to the **data** directory; in our samples – it’s the GR2 file.

See <OutputTagCollection> and <OutputTag> tags above for the description of their attributes and values

For example:

```
<Contents>
    <Content Name="example_tentacle" Type="model">
        <ContentObject Name="" Type="render_model">
            <ContentNetwork Name="default" Type="">
                <InputFile>objects\equipment\example_tentacle\models\example_tentacle.fbx</InputFile>

<IntermediateFile>objects\equipment\example_tentacle\export\models\example_tentacle.gr2</IntermediateFile>
            </ContentNetwork>
            <OutputTagCollection>
                <OutputTag
Type="render_model">objects\equipment\example_tentacle\example_tentacle</OutputTag>
            </OutputTagCollection>
        </ContentObject>
        <ContentObject Name="" Type="skeleton">
            <ContentNetwork Name="default" Type="">

<IntermediateFile>objects\equipment\example_tentacle\export\models\example_tentacle.gr2</IntermediateFile>
                <InputFile>objects\equipment\example_tentacle\models\example_tentacle.fbx</InputFile>
            </ContentNetwork>
            <OutputTagCollection />
        </ContentObject>
        <ContentObject Name="" Type="collision_model">
            <ContentNetwork Name="default" Type="">
                <InputFile>objects\equipment\example_tentacle\models\example_tentacle.fbx</InputFile>

<IntermediateFile>objects\equipment\example_tentacle\export\models\example_tentacle.gr2</IntermediateFile>
            </ContentNetwork>
            <OutputTagCollection>
                <OutputTag
Type="collision_model">objects\equipment\example_tentacle\example_tentacle</OutputTag>
            </OutputTagCollection>
        </ContentObject>
        <ContentObject Name="" Type="physics_model">
            <ContentNetwork Name="default" Type="">
                <InputFile>objects\equipment\example_tentacle\models\example_tentacle.fbx</InputFile>

<IntermediateFile>objects\equipment\example_tentacle\export\models\example_tentacle.gr2</IntermediateFile>
            </ContentNetwork>
            <OutputTagCollection>
                <OutputTag
Type="physics_model">objects\equipment\example_tentacle\example_tentacle</OutputTag>
            </OutputTagCollection>
        </ContentObject>
        <ContentObject Name="" Type="model_animation_graph">
            <ContentNetwork Name="patrol unarmed idle" Type="Base" ModelAnimationMovementData="None">
                <InputFile>objects\equipment\example_tentacle\models\example_tentacle.fbx</InputFile>

<IntermediateFile>objects\equipment\example_tentacle\export\models\example_tentacle.gr2</IntermediateFile>
            </ContentNetwork>
            <OutputTagCollection>
                <OutputTag
Type="frame_event_list">objects\equipment\example_tentacle\example_tentacle</OutputTag>
            <OutputTag
Type="model_animation_graph">objects\equipment\example_tentacle\example_tentacle</OutputTag>
            </OutputTagCollection>
        </ContentObject>
    </Content>
</Contents>
```

Import to Tags

Import to tag files (from GR2) is performed by the **import** command of **tool.exe** or **tool_fast.exe**. However, to perform this import correctly, you need a valid sidecar XML file (see Creation of Sidecar XML file above).

The syntax of the import command is the following:

```
tool_fast.exe import <path to .sidecar.xml> (<in case of level: additional options>)
```

Where:

- <path to .sidecar.xml> – The path to the sidecar XML file (<name_of_your_model>.sidecar.xml), including its name, relative to the data folder.
- <in case of level: additional options> – If you are importing a level to tag files, you will need to specify additional parameters of the import command and perform an additional step after import, see 3.3 for details. For regular models, these parameters are omitted.

For example, for a model:

```
tool_fast.exe import objects\custom_models\example_tentacle\example_tentacle.sidecar.xml
```

OR, for a level:

```
tool_fast.exe import data\levels\custom_levels\example_level\example_level.sidecar.xml
```

After that, all content from the GR2 and its source FBX will be imported to tag files. Particular tag files that will be created are specified in the sidecar XML file, see above.

Additional Properties and Import Steps for Levels

> **NOTE**
> A level can be divided into multiple BSPs (Binary Space Partitions) during its creation in the 3D modelling program. However, most of the custom levels have a single BSP only. Large campaign levels may have multiple BSPs. The BSP concept is out of the scope of this guide.

Moreover, to import the level correctly, you will also need to additionally calculate lighting for it, after execution of the import command.

This is done by the **calc_lm_farm_local.py** python script, which is located at the root directory of the Halo Reach Editing Kit (HREK).

To launch this script, you will need to install Python.

> **NOTE**
> Versions greater than Python 3.6 should be used for launching this script.

> **NOTE**
> To execute python scripts without specifying the full path to them, you will need to add the path to your python executable to the PATH environment variable

The format of the command is the following:

```
python calc_lm_farm_local.py <scenario tag file, w/o .ext> <bsp name> <quality> (<light_group>)
```

Where the parameters are the following:

- **<scenario tag file, w/o .ext>** – The path to the .scenario tag file in this command is specified relative to the tags folder. The name of the .scenario tag file is specified without the file extension. E.g., **levels\mod_levels\my_level_1\my_level_1**
- **<bsp name>** – The name of the BSP (Binary Space Partition) for which the light is calculated. Or, "all" for all BSPs of the level.
- **<quality>** – the quality of the calculated lighting. Please note that the higher the quality is, the more time the calculations do require.

 The possible values here are:

 - **high**
 - **medium**
 - **low**
 - **direct_only**
 - **super_slow**
 - **draft**
 - **debug**
- **<light_group>** – optional parameter, can be omitted. This is the name of the light group, if it is specified, the light will be calculated not for the whole level but for its part.

For example:

```
python calc_lm_farm_local.py levels\example_level\example_level all low
```

Importing Materials (Shaders)

The pipeline necessary to correctly set up materials and shaders for your model i typically the following:

1. When editing a source of the model in the 3d editing software, we specify names for materials. During importing, names of all specified materials will be taken directly from the FBX file.

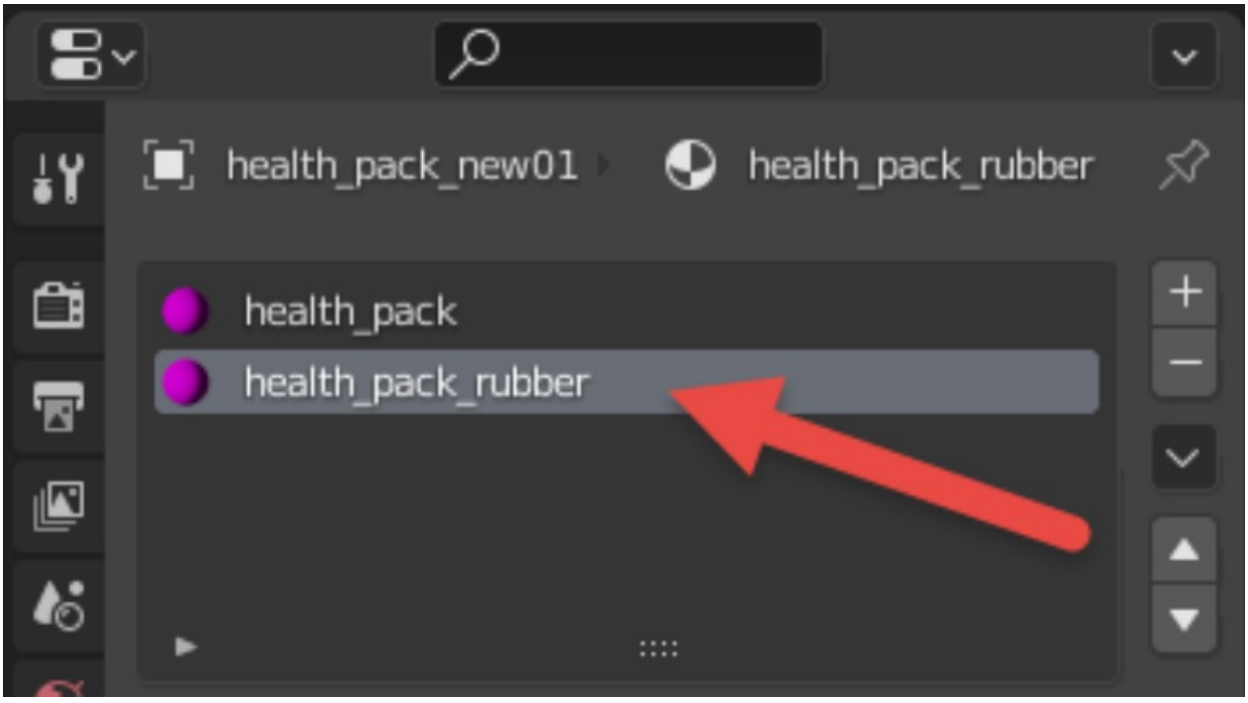


Fig. 3. View of material names within 3D editing software.

2. We create necessary new materials in Foundation in the form of the shader tag files of various types: **.shader**, **.shader_water**, and so on (see the description of the **shader_type** property in Appendix B for the full list of shader types). Or, we locate necessary existing shader tag files that we want to use for our materials.

3. In the JSON file that we prepare before the final import from FBX to GR2, in the "**material_properties**" section of it, we create entries named exactly as materials in the FBX file. And, for every entry, we specify the type of the corresponding shader and the path to it. See Material Properties (the "material_properties" section) for details.

4. Using this JSON file, during importing from FBX to GR2, the system identifies shader tag files that need to be assigned to materials from FBX and sets up the GR2 file correspondingly.

5. Then, we import GR2 with the help of a sidecar XML file. During this import, paths to shader tag files and their types are read from the GR2 and, as a result, the imported model is set up to use these shader tag files.

Appendix A: Samples

In the zip archive with samples, you will find 3 samples that will allow you to test FBX to GR2 conversion.

These samples are the following:

- **Click here for example_health_pack.zip** – the sample model of a health pack, skinned with textures. It has multiple FBX files, acting as sources for render model tag file, physics model tag file, collision model tag file, and markers.

 > **NOTE**
 > Along with other things, this sample also contains the JSON file where properties of the model materials are correctly set up (**example_health_pack_render.json**), see Material Properties (the "material_properties" section) for details.

- **Click here for example_level** – the sample model of a level, with lighting.
- **Click here for example_tentacle** – the sample model with multiple (2) nodes in the source FBX file.

> **NOTE**
> Along with FBX files, these samples also contain valid JSON files needed to convert FBX to GR2 and valid sidecar XML files (<name_of_tag>.sidecar.xml) that are required for import of GR2 files to game engine format.

Appendix B: JSON Parameters

> **NOTE**
> In the Additional Info column below, you will find the following info for some properties:

- **Group of properties** – The group of a property. For example, properties that define the properties of an object are grouped into the "**object level properties**" group, properties that affect appearance are grouped into the "**face level properties**" group, etc. You do not need to use names of these groups anywhere, they are conventional.

- **Valid for object_type** – The particular **object_type** for which this property can be used. For example, some properties can be used only for objects with **object_type** = **_connected_geometry_object_type_mesh** and cannot be used for other objects.
- **Valid for particular subtype only** – The particular subtype of objects for which this property can be used. For example, some properties can be used only for objects with **object_type** = **_connected_geometry_object_type_mesh** where, additionally, the **mesh_primitive_type** is equal to **_connected_geometry_primitive_type_box**, and cannot be used for other objects and other subtypes.

NAME	TYPE	VALUES	ADDITIONAL INFO
object_type	_enum	_connected_geometry_object_type_none; _connected_geometry_object_type_frame; _connected_geometry_object_type_marker; _connected_geometry_object_type_mesh; _connected_geometry_object_type_light; _connected_geometry_object_type_animation_control; _connected_geometry_object_type_animation_camera; _connected_geometry_object_type_animation_event; _connected_geometry_object_type_debug;	The **object_type** is the general property of an object. It is used both for node objects in the "**nodes_properties**" section and, separately, for meshes in "**meshes_properties**". **Group of properties**: object level properties **Valid for object_type**: all
object_animates	_boolean	true/false (0/1)	The **object_animates** is the general property of an object. It is used both for node objects in the "**nodes_properties**" section and, separately, for meshes in "**meshes_properties**". **Group of properties**: object level properties **Valid for object_type**: all
object_ID	_name	GUID(36), for details, see samples.	The **object_ID** is the general property of an object. It is used both for node objects in the "**nodes_properties**" section and, separately, for meshes in "**meshes_properties**". **Group of properties**: object level properties **Valid for object_type**: all
frame_ID1	_integer		**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_frame

NAME	TYPE	VALUES	ADDITIONAL INFO
frame_ID2	_integer		**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_frame
mesh_type	_enum	_connected_geometry_mesh_type_none; _connected_geometry_mesh_type_boundary_surface; _connected_geometry_mesh_type_collision; _connected_geometry_mesh_type_default; _connected_geometry_mesh_type_poop; _connected_geometry_mesh_type_poop_collision; _connected_geometry_mesh_type_poop_physics; _connected_geometry_mesh_type_poop_marker; _connected_geometry_mesh_type_poop_rain_blocker; _connected_geometry_mesh_type_poop_vertical_rain_sheet; _connected_geometry_mesh_type_decorator; _connected_geometry_mesh_type_object_instance; _connected_geometry_mesh_type_physics; _connected_geometry_mesh_type_portal; _connected_geometry_mesh_type_seam; _connected_geometry_mesh_type_planar_fog_volume; _connected_geometry_mesh_type_water_physics_volume; _connected_geometry_mesh_type_water_surface; _connected_geometry_mesh_type_lightmap_region; _connected_geometry_mesh_type_cookie_cutter;	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_mesh
mesh_global_material	_string		**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_mesh

NAME	TYPE	VALUES	ADDITIONAL INFO
mesh_primitive_type	_enum	_connected_geometry_primitive_type_none; _connected_geometry_primitive_type_box; _connected_geometry_primitive_type_pill; _connected_geometry_primitive_type_sphere;	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_mesh
mesh_tessellation_density	_enum	_connected_geometry_mesh_tessellation_density_none; _connected_geometry_mesh_tessellation_density_4x; _connected_geometry_mesh_tessellation_density_9x; _connected_geometry_mesh_tessellation_density_36x;	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_mesh
mesh_additional_compression	_enum	_connected_geometry_mesh_additional_compression_force_on; _connected_geometry_mesh_additional_compression_force_off; _connected_geometry_mesh_additional_compression_default;	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_mesh
mesh_primitive_box_length	_number	0.0	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_mesh **Valid for particular subtype only**: PRIMITIVE(box), i.e. mesh_primitive_type = _connected_geometry_primitive_type_box;
mesh_primitive_box_width	_number	0.0	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_mesh **Valid for particular subtype only**: PRIMITIVE(box), i.e. mesh_primitive_type = _connected_geometry_primitive_type_box;

NAME	TYPE	VALUES	ADDITIONAL INFO
mesh_primitive_box_height	_number	0.0	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_mesh **Valid for particular subtype only**: PRIMITIVE(box), i.e. mesh_primitive_type = _connected_geometry_primitive_type_box;
mesh_primitive_pill_radius	_number	0.0	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_mesh **Valid for particular subtype only**: PRIMITIVE(box), i.e. mesh_primitive_type = _connected_geometry_primitive_type_pill;
mesh_primitive_pill_height	_number	0.0	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_mesh **Valid for particular subtype only**: PRIMITIVE(box), i.e. mesh_primitive_type = _connected_geometry_primitive_type_pill;
mesh_primitive_sphere_radius	_number	0.0	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_mesh **Valid for particular subtype only**: PRIMITIVE(box), i.e. mesh_primitive_type = _connected_geometry_primitive_type_sphere;
mesh_boundary_surface_type	_enum	_connected_geometry_boundary_surface_type_none; _connected_geometry_boundary_surface_type_soft_ceiling; _connected_geometry_boundary_surface_type_soft_kill; _connected_geometry_boundary_surface_type_slip_surface;	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_mesh **Valid for particular subtype only**: MESH_TYPE(boundary_surface), i.e. mesh_type = _connected_geometry_mesh_type_boundary_surface;

NAME	TYPE	VALUES	ADDITIONAL INFO
mesh_boundary_surface_name	_name	32 characters	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_mesh **Valid for particular subtype only**: MESH_TYPE(boundary_surface), i.e. mesh_type = _connected_geometry_mesh_type_boundary_surface;
mesh_poop_lighting	_enum	_connected_geometry_poop_lighting_none; _connected_geometry_poop_lighting_single_probe; _connected_geometry_poop_lighting_per_pixel; _connected_geometry_poop_lighting_per_vertex;	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_mesh **Valid for particular subtype only**: MESH_TYPE(poop), i.e. mesh_type = _connected_geometry_mesh_type_poop;
mesh_poop_pathfinding	_enum	_connected_poop_instance_pathfinding_policy_none; _connected_poop_instance_pathfinding_policy_cutout; _connected_poop_instance_pathfinding_policy_static;	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_mesh **Valid for particular subtype only**: MESH_TYPE(poop), i.e. mesh_type = _connected_geometry_mesh_type_poop;
mesh_poop_imposter_policy	_enum	*connected_poop_instance_imposter_policy_polygon_default;*; _connected_poop_instance_imposter_policy_polygon_high; _connected_poop_instance_imposter_policy_card_default; _connected_poop_instance_imposter_policy_card_high; _connected_poop_instance_imposter_policy_none; _connected_poop_instance_imposter_policy_never;	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_mesh **Valid for particular subtype only**: MESH_TYPE(poop), i.e. mesh_type = _connected_geometry_mesh_type_poop;

NAME	TYPE	VALUES	ADDITIONAL INFO
mesh_poop_imposter_transition_distance	_number	-1.0	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_mesh **Valid for particular subtype only**: MESH_TYPE(poop), i.e. mesh_type = _connected_geometry_mesh_type_poop;
mesh_poop_fade_range_start	_number	36.0	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_mesh **Valid for particular subtype only**: MESH_TYPE(poop), i.e. mesh_type = _connected_geometry_mesh_type_poop;
mesh_poop_fade_range_end	_number	30.0	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_mesh **Valid for particular subtype only**: MESH_TYPE(poop), i.e. mesh_type = _connected_geometry_mesh_type_poop;
mesh_poop_is_render_only	_boolean	true/false	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_mesh **Valid for particular subtype only**: MESH_TYPE(poop), i.e. mesh_type = _connected_geometry_mesh_type_poop;
mesh_poop_chops_portals	_boolean	true/false	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_mesh **Valid for particular subtype only**: MESH_TYPE(poop), i.e. mesh_type = _connected_geometry_mesh_type_poop;

NAME	TYPE	VALUES	ADDITIONAL INFO
mesh_poop_does_not_block_aoe	_boolean	true/false	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_mesh **Valid for particular subtype only**: MESH_TYPE(poop), i.e. mesh_type = _connected_geometry_mesh_type_poop;
mesh_poop_excluded_from_lightprobes	_boolean	true/false	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_mesh **Valid for particular subtype only**: MESH_TYPE(poop), i.e. mesh_type = _connected_geometry_mesh_type_poop;
mesh_poop_decal_spacing	_boolean	true/false	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_mesh **Valid for particular subtype only**: MESH_TYPE(poop), i.e. mesh_type = _connected_geometry_mesh_type_poop;
mesh_poop_precise_geometry	_boolean	true/false	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_mesh **Valid for particular subtype only**: MESH_TYPE(poop), i.e. mesh_type = _connected_geometry_mesh_type_poop;
mesh_poop_decomposition_hulls	_integer	-1	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_mesh **Valid for particular subtype only**: MESH_TYPE(poop), i.e. mesh_type = _connected_geometry_mesh_type_poop;

NAME	TYPE	VALUES	ADDITIONAL INFO
mesh_poop_predominant_shader_name	_name	1024 characters	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_mesh **Valid for particular subtype only**: MESH_TYPE(poop), i.e. mesh_type = _connected_geometry_mesh_type_poop;
mesh_portal_type	_enum	_connected_geometry_portal_type_none; _connected_geometry_portal_type_no_way; _connected_geometry_portal_type_one_way; _connected_geometry_portal_type_two_way	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_mesh **Valid for particular subtype only**: MESH_TYPE(portal), i.e. mesh_type = _connected_geometry_mesh_type_portal;
mesh_portal_ai_deafening	_boolean	true/false	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_mesh **Valid for particular subtype only**: MESH_TYPE(portal), i.e. mesh_type = _connected_geometry_mesh_type_portal;
mesh_portal_blocks_sound	_boolean	true/false	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_mesh **Valid for particular subtype only**: MESH_TYPE(portal), i.e. mesh_type = _connected_geometry_mesh_type_portal;
mesh_portal_is_door	_boolean	true/false	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_mesh **Valid for particular subtype only**: MESH_TYPE(portal), i.e. mesh_type = _connected_geometry_mesh_type_portal;

NAME	TYPE	VALUES	ADDITIONAL INFO
mesh_decorator_lod	_integer	1	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_mesh
mesh_decorator_name	_name	32 characters	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_mesh
mesh_seam_associated_bsp	_name	32 characters	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_mesh **Valid for particular subtype only**: MESH_TYPE(seam), i.e. mesh_type = _connected_geometry_mesh_type_seam;
mesh_water_volume_depth	_number	20.0	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_mesh **Valid for particular subtype only**: MESH_TYPE(water_physics_volume), i.e. mesh_type = _connected_geometry_mesh_type_water_physics_volume;
mesh_water_volume_flow_direction	_number	0.0	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_mesh **Valid for particular subtype only**: MESH_TYPE(water_physics_volume), i.e. mesh_type = _connected_geometry_mesh_type_water_physics_volume;

NAME	TYPE	VALUES	ADDITIONAL INFO
mesh_water_volume_flow_velocity	_number	0.0	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_mesh **Valid for particular subtype only**: MESH_TYPE(water_physics_volume), i.e. mesh_type = _connected_geometry_mesh_type_water_physics_volume;
mesh_water_volume_fog_color	_vector	0	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_mesh **Valid for particular subtype only**: MESH_TYPE(water_physics_volume), i.e. mesh_type = _connected_geometry_mesh_type_water_physics_volume;
mesh_water_volume_fog_murkiness	_number	0.0	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_mesh **Valid for particular subtype only**: MESH_TYPE(water_physics_volume), i.e. mesh_type = _connected_geometry_mesh_type_water_physics_volume;
mesh_fog_name	_name	32 characters	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_mesh **Valid for particular subtype only**: MESH_TYPE(planar_fog_volume), i.e. mesh_type = _connected_geometry_mesh_type_planar_fog_volume;

NAME	TYPE	VALUES	ADDITIONAL INFO
mesh_fog_volume_depth	_number	20.0	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_mesh **Valid for particular subtype only**: MESH_TYPE(planar_fog_volume), i.e. mesh_type = _connected_geometry_mesh_type_planar_fog_volume;
marker_type	_enum	_connected_geometry_marker_type_none; _connected_geometry_marker_type_game_instance; _connected_geometry_marker_type_model; _connected_geometry_marker_type_pathfinding_sphere; _connected_geometry_marker_type_water_volume_flow; _connected_geometry_marker_type_physics_hinge_constraint; _connected_geometry_marker_type_physics_socket_constraint; _connected_geometry_marker_type_target; _connected_geometry_marker_type_garbage; _connected_geometry_marker_type_effects; _connected_geometry_marker_type_hint;	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_marker;
marker_region	_string	connected_geometry_regions_table	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_marker; **Valid for particular subtype only**: mesh_type = _connected_geometry_mesh_type_object_instance
marker_all_regions	_boolean	true/false	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_marker; **Valid for particular subtype only**: mesh_type = _connected_geometry_mesh_type_object_instance

NAME	TYPE	VALUES	ADDITIONAL INFO
marker_game_instance_tag_name	_name	64 characters	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_marker; **Valid for particular subtype only**: MARKER_TYPE(game_instance), i.e.marker_type =_connected_geometry_marker_type_game_instance;
marker_game_instance_variant_name	_name	64 characters	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_marker; **Valid for particular subtype only**: MARKER_TYPE(game_instance), i.e.marker_type =_connected_geometry_marker_type_game_instance;
marker_model_group	_name	64 characters	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_marker; **Valid for particular subtype only**: MARKER_TYPE(model), i.e. marker_type = _connected_geometry_marker_type_model; MARKER_TYPE(hint), i.e. marker_type = _connected_geometry_marker_type_hint; MARKER_TYPE(target), i.e. marker_type = _connected_geometry_marker_type_target;
marker_velocity	_vector	0	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_marker; **Valid for particular subtype only**: MARKER_TYPE(model), i.e. marker_type = _connected_geometry_marker_type_model;

NAME	TYPE	VALUES	ADDITIONAL INFO
marker_pathfinding_sphere_vehicle_only	_boolean	true/false	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_marker; **Valid for particular subtype only**: MARKER_TYPE(pathfinding_sphere), i.e. marker_type = _connected_geometry_marker_type_pathfinding_sphere;
marker_pathfinding_sphere_remains_when_open	_boolean	true/false	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_marker; **Valid for particular subtype only**: MARKER_TYPE(pathfinding_sphere), i.e. marker_type = _connected_geometry_marker_type_pathfinding_sphere;
marker_pathfinding_sphere_with_sectors	_boolean	true/false	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_marker; **Valid for particular subtype only**: MARKER_TYPE(pathfinding_sphere), i.e. marker_type = _connected_geometry_marker_type_pathfinding_sphere;
physics_constraint_parent	_name		**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_marker; **Valid for particular subtype only**: MARKER_TYPE(physics_hinge_constraint), i.e. marker_type = _connected_geometry_marker_type_physics_hinge_constraint; MARKER_TYPE(physics_socket_constraint), i.e. marker_type = _connected_geometry_marker_type_physics_socket_constraint;

NAME	TYPE	VALUES	ADDITIONAL INFO
physics_constraint_child	_name		**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_marker; **Valid for particular subtype only**: MARKER_TYPE(physics_hinge_constraint), i.e. marker_type = _connected_geometry_marker_type_physics_hinge_constraint; MARKER_TYPE(physics_socket_constraint), i.e. marker_type = _connected_geometry_marker_type_physics_socket_constraint;
physics_constraint_use_limits	_boolean	true/false	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_marker; **Valid for particular subtype only**: MARKER_TYPE(physics_hinge_constraint), i.e. marker_type = _connected_geometry_marker_type_physics_hinge_constraint; MARKER_TYPE(physics_socket_constraint), i.e. marker_type = _connected_geometry_marker_type_physics_socket_constraint;
physics_constraint_hinge_min	_number	-180.0	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_marker; **Valid for particular subtype only**: MARKER_TYPE(physics_hinge_constraint), i.e. marker_type = _connected_geometry_marker_type_physics_hinge_constraint;

NAME	TYPE	VALUES	ADDITIONAL INFO
physics_constraint_hinge_max	_number	180.0	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_marker; **Valid for particular subtype only**: MARKER_TYPE(physics_hinge_constraint), i.e. marker_type = _connected_geometry_marker_type_physics_hinge_constraint;
physics_constraint_cone_angle	_number	90.0	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_marker; **Valid for particular subtype only**: MARKER_TYPE(physics_socket_constraint), i.e. marker_type = _connected_geometry_marker_type_physics_socket_constraint;
physics_constraint_plane_min	_number	-90.0	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_marker; **Valid for particular subtype only**: MARKER_TYPE(physics_socket_constraint), i.e. marker_type = _connected_geometry_marker_type_physics_socket_constraint;
physics_constraint_plane_max	_number	90.0	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_marker; **Valid for particular subtype only**: MARKER_TYPE(physics_socket_constraint), i.e. marker_type = _connected_geometry_marker_type_physics_socket_constraint;

NAME	TYPE	VALUES	ADDITIONAL INFO
physics_constraint_twist_start	_number	-180.0	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_marker; **Valid for particular subtype only**: MARKER_TYPE(physics_socket_constraint), i.e. marker_type = _connected_geometry_marker_type_physics_socket_constraint;
physics_constraint_twist_end	_number	180.0	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_marker; **Valid for particular subtype only**: MARKER_TYPE(physics_socket_constraint), i.e. marker_type = _connected_geometry_marker_type_physics_socket_constraint;
animation_control_type	_enum	_connected_geometry_animation_control_type_none; _connected_geometry_animation_control_type_ik_effector ; _connected_geometry_animation_control_type_aim_yaw ; _connected_geometry_animation_control_type_aim_pitch; _connected_geometry_animation_control_type_orient_constraint; _connected_geometry_animation_control_type_target_proxy; _connected_geometry_animation_control_type_ik_pole_v	
ector;	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_animation_control;		

NAME	TYPE	VALUES	ADDITIONAL INFO
animation_control_id	_integer	0	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_animation_control;
animation_control_ik_chain	_name		**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_animation_control;
animation_control_ik_effect	_number	0.0	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_animation_control;
animation_control_constraint_effect	_number	0.0	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_animation_control;
animation_control_proxy_target_marker	_name		**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_animation_control;
animation_control_proxy_target_tag	_name		**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_animation_control;
animation_control_proxy_target_usage	_name		**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_animation_control;

NAME	TYPE	VALUES	ADDITIONAL INFO
animation_event_type	_enum	_connected_geometry_animation_event_type_none; _connected_geometry_animation_event_type_custom; _connected_geometry_animation_event_type_loop; _connected_geometry_animation_event_type_sound; _connected_geometry_animation_event_type_effect; _connected_geometry_animation_event_type_ik_active; _connected_geometry_animation_event_type_ik_passive; _connected_geometry_animation_event_type_text; _connected_geometry_animation_event_type_wrinkle_map; _connected_geometry_animation_event_type_footstep; _connected_geometry_animation_event_type_cinematic_effect; _connected_geometry_animation_event_type_object_function; _connected_geometry_animation_event_type_frame; _connected_geometry_animation_event_type_import;	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_animation_event;
animation_event_start	_number	0.0	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_animation_event;
animation_event_end	_number	0.0	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_animation_event;
animation_event_id	_integer	0	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_animation_event;
animation_event_wrinkle_map_face_region	_name		**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_animation_event;

NAME	TYPE	VALUES	ADDITIONAL INFO
animation_event_wrinkle_map_effect	_number	0.0	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_animation_event;
animation_event_footstep_type	_name		**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_animation_event;
animation_event_footstep_effect	_number	0.0	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_animation_event;
animation_event_ik_chain	_name		**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_animation_event;
animation_event_ik_active_tag	_name		**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_animation_event;
animation_event_ik_target_tag	_name		**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_animation_event;
animation_event_ik_target_marker	_name		**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_animation_event;
animation_event_ik_target_usage	_name		**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_animation_event;
animation_event_ik_proxy_target_id	_integer	0	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_animation_event;

NAME	TYPE	VALUES	ADDITIONAL INFO
animation_event_ik_pole_vector_id	_integer	0	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_animation_event;
animation_event_ik_effector_id	_integer	0	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_animation_event;
animation_event_cinematic_effect_tag	_name		**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_animation_event;
animation_event_cinematic_effect_effect	_number	0.0	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_animation_event;
animation_event_cinematic_effect_marker	_name		**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_animation_event;
animation_event_object_function_name	_name		**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_animation_event;
animation_event_object_function_effect	_number	0.0	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_animation_event;
animation_event_frame_frame	_integer	0	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_animation_event;
animation_event_frame_name	_name		**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_animation_event;

NAME	TYPE	VALUES	ADDITIONAL INFO
animation_event_frame_trigger	_boolen	true/false	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_animation_event;
animation_event_import_frame	_integer	0	**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_animation_event;
animation_event_import_name	_name		**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_animation_event;
animation_event_text	_name		**Group of properties**: object level properties **Valid for object_type**: _connected_geometry_object_type_animation_event;
face_type	_enum	_connected_geometry_face_type_none; _connected_geometry_face_type_normal; _connected_geometry_face_type_seam_sealer; _connected_geometry_face_type_sky; _connected_geometry_face_type_weather_polyhedra;	**Group of properties**: face level properties **Valid for object_type**: _connected_geometry_object_type_mesh;
face_mode	_enum	_connected_geometry_face_mode_none; _connected_geometry_face_mode_normal; _connected_geometry_face_mode_render_only; _connected_geometry_face_mode_collision_only; _connected_geometry_face_mode_sphere_collision_only; _connected_geometry_face_mode_shadow_only; _connected_geometry_face_mode_lightmap_only; _connected_geometry_face_mode_breakable;	**Group of properties**: face level properties **Valid for object_type**: _connected_geometry_object_type_mesh;

NAME	TYPE	VALUES	ADDITIONAL INFO
face_sides	_enum	_connected_geometry_face_sides_none; _connected_geometry_face_sides_one_sided; _connected_geometry_face_sides_one_sided_transparent; _connected_geometry_face_sides_two_sided; _connected_geometry_face_sides_two_sided_transparent;	**Group of properties**: face level properties **Valid for object_type**: _connected_geometry_object_type_mesh;
face_draw_distance	_enum	_connected_geometry_face_draw_distance_none; _connected_geometry_face_draw_distance_normal; _connected_geometry_face_draw_distance_detail_mid; _connected_geometry_face_draw_distance_detail_mid;	**Group of properties**: face level properties **Valid for object_type**: _connected_geometry_object_type_mesh;
face_global_material	_string	connected_geometry_global_material_table	**Group of properties**: face level properties **Valid for object_type**: _connected_geometry_object_type_mesh;
face_region	_string	connected_geometry_regions_table	**Group of properties**: face level properties **Valid for object_type**: _connected_geometry_object_type_mesh;
texcoord_usage	_enum	_connected_material_texcoord_usage_default; _connected_material_texcoord_usage_none; _connected_material_texcoord_usage_anisotropic;	**Group of properties**: face level properties **Valid for object_type**: _connected_geometry_object_type_mesh;
conveyor	_boolean	true/false	**Group of properties**: face level properties **Valid for object_type**: _connected_geometry_object_type_mesh;
ladder	_boolean	true/false	**Group of properties**: face level properties **Valid for object_type**: _connected_geometry_object_type_mesh;
slip_surface	_boolean	true/false	**Group of properties**: face level properties **Valid for object_type**: _connected_geometry_object_type_mesh;

NAME	TYPE	VALUES	ADDITIONAL INFO
decal_offset	_boolean	true/false	**Group of properties**: face level properties **Valid for object_type**: _connected_geometry_object_type_mesh;
group_transparents_by_plane	_boolean	true/false	**Group of properties**: face level properties **Valid for object_type**: _connected_geometry_object_type_mesh;
no_shadow	_boolean	true/false	**Group of properties**: face level properties **Valid for object_type**: _connected_geometry_object_type_mesh;
precise_position	_boolean	true/false	**Group of properties**: face level properties **Valid for object_type**: _connected_geometry_object_type_mesh;
sky_permutation_index	_integer	0	**Group of properties**: face level properties **Valid for object_type**: _connected_geometry_object_type_mesh; **Valid for particular subtype only**: face_type = _connected_geometry_face_type_sky;
lightmap_additive_transparency	_vector	0	**Group of properties**: lightmap configuration properties **Valid for object_type**: _connected_geometry_object_type_mesh;
lightmap_ignore_default_resolution_scale	_boolean	true/false	**Group of properties**: lightmap configuration properties **Valid for object_type**: _connected_geometry_object_type_mesh;
lightmap_resolution_scale	_integer	3	**Group of properties**: lightmap configuration properties **Valid for object_type**: _connected_geometry_object_type_mesh;

NAME	TYPE	VALUES	ADDITIONAL INFO
lightmap_chart_group	_integer	0	**Group of properties**: lightmap configuration properties **Valid for object_type**: _connected_geometry_object_type_mesh;
lightmap_type	_enum	_connected_material_lightmap_type_per_pixel; _connected_material_lightmap_type_per_vertex;	**Group of properties**: lightmap configuration properties **Valid for object_type**: _connected_geometry_object_type_mesh;
lightmap_transparency_override	_boolean	true/false	**Group of properties**: lightmap configuration properties **Valid for object_type**: _connected_geometry_object_type_mesh;
lightmap_analytical_bounce_modifier	_number	9999.0	**Group of properties**: lightmap configuration properties **Valid for object_type**: _connected_geometry_object_type_mesh;
lightmap_general_bounce_modifier	_nuber	9999.0	**Group of properties**: lightmap configuration properties **Valid for object_type**: _connected_geometry_object_type_mesh;
lightmap_translucency_tint_color	_vector	0	**Group of properties**: lightmap configuration properties **Valid for object_type**: _connected_geometry_object_type_mesh;
lightmap_lighting_from_both_sides	_boolean	true/false	**Group of properties**: lightmap configuration properties **Valid for object_type**: _connected_geometry_object_type_mesh;
lighting_attenuation_cutoff	_number	0.0	**Group of properties**: material lighting properties **Valid for object_type**: _connected_geometry_object_type_mesh;

NAME	TYPE	VALUES	ADDITIONAL INFO
lighting_attenuation_falloff	_number	0.0	**Group of properties**: material lighting properties **Valid for object_type**: _connected_geometry_object_type_mesh;
lighting_emissive_focus	_number	0.0	**Group of properties**: material lighting properties **Valid for object_type**: _connected_geometry_object_type_mesh;
lighting_emissive_color	_vector	0xFFFFFFFF	**Group of properties**: material lighting properties **Valid for object_type**: _connected_geometry_object_type_mesh;
lighting_emissive_per_unit	_boolean	true/false	**Group of properties**: material lighting properties **Valid for object_type**: _connected_geometry_object_type_mesh;
lighting_emissive_power	_number	0.0	**Group of properties**: material lighting properties **Valid for object_type**: _connected_geometry_object_type_mesh;
lighting_emissive_quality	_number	1.0	**Group of properties**: material lighting properties **Valid for object_type**: _connected_geometry_object_type_mesh;
lighting_use_shader_gel	_boolean	true/false	**Group of properties**: material lighting properties **Valid for object_type**: _connected_geometry_object_type_mesh;
lighting_bounce_ratio	_number	1.0	**Group of properties**: material lighting properties **Valid for object_type**: _connected_geometry_object_type_mesh;
light_type	_enum	_connected_geometry_light_type_omni; _connected_geometry_light_type_spot; _connected_geometry_light_type_directional;	**Group of properties**: lightmapper placed lights properties **Valid for object_type**: _connected_geometry_object_type_light;

NAME	TYPE	VALUES	ADDITIONAL INFO
light_type_version	_number	0.0	**Group of properties**: lightmapper placed lights properties **Valid for object_type**: _connected_geometry_object_type_light;
light_game_type	_enum	_connected_geometry_bungie_light_type_default; _connected_geometry_bungie_light_type_uber; _connected_geometry_bungie_light_type_inlined; _connected_geometry_bungie_light_type_screen_space; _connected_geometry_bungie_light_type_rerender;	**Group of properties**: lightmapper placed lights properties **Valid for object_type**: _connected_geometry_object_type_light;
light_shape	_enum	_connected_geometry_light_shape_rectangle; _connected_geometry_light_shape_circle;	**Group of properties**: lightmapper placed lights properties **Valid for object_type**: _connected_geometry_object_type_light;
light_use_near_attenuation	_boolean	true/false	**Group of properties**: lightmapper placed lights properties **Valid for object_type**: _connected_geometry_object_type_light;
light_near_attenuation_start	_number	0.0	**Group of properties**: lightmapper placed lights properties **Valid for object_type**: _connected_geometry_object_type_light;
light_near_attenuation_end	_number	0.0	**Group of properties**: lightmapper placed lights properties **Valid for object_type**: _connected_geometry_object_type_light;
light_use_far_attenuation	_boolean	true/false	**Group of properties**: lightmapper placed lights properties **Valid for object_type**: _connected_geometry_object_type_light;

NAME	TYPE	VALUES	ADDITIONAL INFO
light_volume_distance	_number	0.0	**Group of properties**: lightmapper placed lights properties **Valid for object_type**: _connected_geometry_object_type_light;
light_volume_intensity_scalar	_number	1.0	**Group of properties**: lightmapper placed lights properties **Valid for object_type**: _connected_geometry_object_type_light;
light_fade_start_distance	_number	100.0	**Group of properties**: lightmapper placed lights properties **Valid for object_type**: _connected_geometry_object_type_light;
light_fade_out_distance	_number	150.0	**Group of properties**: lightmapper placed lights properties **Valid for object_type**: _connected_geometry_object_type_light;
light_ignore_bsp_visibility	_boolean	true/false	**Group of properties**: lightmapper placed lights properties **Valid for object_type**: _connected_geometry_object_type_light;
light_far_attenuation_start	_number	0.0	**Group of properties**: lightmapper placed lights properties **Valid for object_type**: _connected_geometry_object_type_light;
light_far_attenuation_end	_number	0.0	**Group of properties**: lightmapper placed lights properties **Valid for object_type**: _connected_geometry_object_type_light;
light_color	_vector	0	**Group of properties**: lightmapper placed lights properties **Valid for object_type**: _connected_geometry_object_type_light;

NAME	TYPE	VALUES	ADDITIONAL INFO
light_intensity	_number	0.0	**Group of properties**: lightmapper placed lights properties **Valid for object_type**: _connected_geometry_object_type_light;
light_clipping_size_x_pos	_number	100.0	**Group of properties**: lightmapper placed lights properties **Valid for object_type**: _connected_geometry_object_type_light;
light_clipping_size_y_pos	_number	100.0	**Group of properties**: lightmapper placed lights properties **Valid for object_type**: _connected_geometry_object_type_light;
light_clipping_size_z_pos	_number	100.0	**Group of properties**: lightmapper placed lights properties **Valid for object_type**: _connected_geometry_object_type_light;
light_clipping_size_x_neg	_number	-100.0	**Group of properties**: lightmapper placed lights properties **Valid for object_type**: _connected_geometry_object_type_light;
light_clipping_size_y_neg	_number	-100.0	**Group of properties**: lightmapper placed lights properties **Valid for object_type**: _connected_geometry_object_type_light;
light_clipping_size_z_neg	_number	-100.0	**Group of properties**: lightmapper placed lights properties **Valid for object_type**: _connected_geometry_object_type_light;
light_use_clipping	_boolean	true/false	**Group of properties**: lightmapper placed lights properties **Valid for object_type**: _connected_geometry_object_type_light;

NAME	TYPE	VALUES	ADDITIONAL INFO
light_hotspot_size	_number	30.0	**Group of properties**: lightmapper placed lights properties **Valid for object_type**: _connected_geometry_object_type_light;
light_hotspot_falloff	_number	45.0	**Group of properties**: lightmapper placed lights properties **Valid for object_type**: _connected_geometry_object_type_light;
light_falloff_shape	_number	1.0	**Group of properties**: lightmapper placed lights properties **Valid for object_type**: _connected_geometry_object_type_light;
light_aspect	_number	1.0	**Group of properties**: lightmapper placed lights properties **Valid for object_type**: _connected_geometry_object_type_light;
light_frustum_width	_number	1.0	**Group of properties**: lightmapper placed lights properties **Valid for object_type**: _connected_geometry_object_type_light;
light_frustum_height	_number	1.0	**Group of properties**: lightmapper placed lights properties **Valid for object_type**: _connected_geometry_object_type_light;
light_bounce_light_ratio	_number	1.0	**Group of properties**: lightmapper placed lights properties **Valid for object_type**: _connected_geometry_object_type_light;
light_dynamic_light_has_bounce	_boolean	true/false	**Group of properties**: lightmapper placed lights properties **Valid for object_type**: _connected_geometry_object_type_light;

NAME	TYPE	VALUES	ADDITIONAL INFO
light_screenspace_light_has_specular	_boolean	true/false	**Group of properties**: lightmapper placed lights properties **Valid for object_type**: _connected_geometry_object_type_light;
light_light_tag_override	_name	128 characters	**Group of properties**: lightmapper placed lights properties **Valid for object_type**: _connected_geometry_object_type_light;
light_shader_reference	_name	128 characters	**Group of properties**: lightmapper placed lights properties **Valid for object_type**: _connected_geometry_object_type_light;
light_gel_reference	_name	128 characters	**Group of properties**: lightmapper placed lights properties **Valid for object_type**: _connected_geometry_object_type_light;
light_lens_flare_reference	_name	128 characters	**Group of properties**: lightmapper placed lights properties **Valid for object_type**: _connected_geometry_object_type_light;
animation_test_property_1	_number	0.0	**Group of properties**: per-node animation properties **Valid for object_type**: _connected_geometry_object_type_none;
is_replacement_correction_node	_boolean	true/false	**Group of properties**: per-node animation properties **Valid for object_type**: _connected_geometry_object_type_none;
is_fik_anchor_node	_boolean	true/false	**Group of properties**: per-node animation properties **Valid for object_type**: _connected_geometry_object_type_none;

NAME	TYPE	VALUES	ADDITIONAL INFO
is_object_space_offset_node	_boolean	true/false	**Group of properties**: per-node animation properties **Valid for object_type**: _connected_geometry_object_type_none;

NAME	TYPE	VALUES	ADDITIONAL INFO
shader_type	_enum	override shader shader_cortana shader_custom shader_decal shader_foliage shader_fur shader_fur_stencil shader_glass shader_halogram shader_mux shader_mux_material shader_screen shader_skin shader_terrain shader_water	**Group of properties**: material properties Valid for the material entry that is specified in the "material_properties" section, see Material Properties (the "material_properties" section). **NOTE**: Values of the **shader_type** mostly correspond to the type of the shader tag file that is created in Foundation. For example, **shader_water**– corresponds to the shader tag file with the .**shader_water** extension. However, along with values corresponding to these types, there is also a special **override** value. If you specify **override** as a value of **shader_type**, you will be able to override properties of the mesh (of its surface covered with this material) directly in the properties of the material by assigning a special value to **shader_path**. Particularly, in this case, as a value of **shader_path** you can pass a mesh property with its new value, in the "*<name_of_mesh_property>=<value_of_this_property>*" format. For example, if you want to set up a surface of a mesh as a sky, you can specify the following in the properties of this material: "sky_material": { "bungie_shader_type" : "override", "bungie_shader_path" : "bungie_face_type=_connected_geometry_face_type_sky" }

NAME	TYPE	VALUES	ADDITIONAL INFO
shader_path	_string		**Group of properties**: material properties Valid for the material entry specified in the "material_properties" section, see Material Properties (the "material_properties" section). **Value:** Value: - a path to the shader tag file (of the specified **shader_type**), relative to the **tags** folder, without file extension, and with double backslashes as path delimiters. For example: "objects\equipment\health_pack\shaders\health_pack_rubber" - or, a special value, overriding properties of the mesh (of its surface covered with this material), when **shader_type** is equal to **override** (see **shader_type** above)

Halo 4 Mod Documentation Coming Soon

11/23/2022 • 2 minutes to read

Known Issues

11/23/2022 • 2 minutes to read

In this article you can find known issues across various mod tools for Halo: The Master Chief Collection. Issues are sorted by Tool and Build version.

Use the links below to jump to a specific Title or Tool.

EXCESSION TOOL	HCE-EK	H2-EK	H2A-EK	H3-EK	H3ODST-EK	HR-EK	H4-EK
-	HCE-Sapien	H2-Sapien	H2A-Sapien	H3-Sapien	H3ODST-Sapien	HR-Sapien	H4-Sapien
-	HCE-Guerilla	H2-Guerilla	H2A-Foundation	H3-Guerilla	H3ODST-Guerilla	HR-Foundation	H4-Foundation

Excession Tool

Build 2022.11.09.167740.5

Max Team Issues

- Changes to the Max Team Category is not being respected in-game.
- HCE Only: Excession will crash after saving any changes made to the "Max Teams All Up" or "Max Team Category" options.
- "Max Team Category" fields can only be edited when the "Max Team All Up" field is set to 0.
- If a mode has the "Max Team Category" set to 0 it can load into a black screen and immediately announce game over.

Halo CE Editor Kit

- N/A

Halo CE Sapien

- N/A

Halo CE Guerilla

- N/A

Halo 2 Editor Kit

- N/A

Halo 2 Sapien

- N/A

Halo 2 Guerilla

- N/A

Halo 2A Editor Kit

- N/A

Halo 2A Sapien

- N/A

Halo 2A Foundation

- N/A

Halo 3 Editor Kit

- N/A

Halo 3 Sapien

- N/A

Halo 3 Guerilla

- N/A

Halo 3 ODST Editor Kit

- N/A

Halo 3 ODST Sapien

- N/A

Halo 3 ODST Guerilla

- N/A

Halo Reach Editor Kit

- N/A

Halo Reach Sapien

- N/A

Halo Reach Foundation

- N/A

Halo 4 Editor Kit

- N/A

Halo 4 Sapien

- N/A

Halo 4 Foundation

- N/A